MONARCH MAYFLY OPTIMIZATION: A NOVEL HYBRID METAHEURISTIC ALGORITHM TO FEATURE SELECTION PROBLEM

A Thesis Presented to the Faculty of Computer Science Department of South Philippine Adventist College, Camanchiles, Matanao, Davao del Sur In Partial Fulfilment of the Requirement for the Degree

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

GARY LOUIS P. GARCIA

MAY 2024

THESIS ABSTRACT

GARY LOUIS P. GARCIA, Bachelor of Science in Computer Science,
South Philippine Adventist College, Camanchiles, Matanao, Davao del Sur, 2024.
"MONARCH MAYFLY OPTIMIZATION: A NOVEL HYBRID METAHEURISTIC ALGORITHM TO FEATURE SELECTION PROBLEM"

Research Adviser: MS. NEILA M. PAGLINAWAN-MUÑEZ, MSCS

Large and expanding datasets often contain irrelevant and redundant features that degrade classifier performance in real-world applications such as image processing, finance, and medicine. This issue can be addressed through feature selection, which aims to maintain high classifier performance with a minimal number of features. To this end, a novel hybrid metaheuristic feature selection algorithm called Monarch Mayfly Optimization (MMO) was developed, combining Monarch Butterfly Optimization (MBO) and the Mayfly algorithm (MA). MMO was evaluated using K-Nearest Neighbours (KNN) and Support Vector Machines (SVM) classifiers across 18 UCI benchmarking datasets. The analysis demonstrated that MMO consistently improves classification accuracy by optimizing feature subsets, outperforming standard classifiers. Comparisons with the MA-HS algorithm revealed that while MA-HS selects fewer features, MMO achieves higher accuracy. This highlights the importance of prioritizing quality and relevance over quantity in feature selection. Additionally, the MMO algorithm exhibits an approximate time complexity of $O(N\ log\ N\ + Classifier_{timeComplexity})$.

ACKNOWLEDGEMENT

The completion of this study would not have been possible without the genuine support of the following individuals. The researcher is profoundly grateful to his parents for their unwavering encouragement, heartfelt prayers, and invaluable financial support, which have been the cornerstone of his perseverance and determination throughout this journey. The researcher extends heartfelt gratitude to his instructor, Ms. Mauren Banasing, whose insightful ideas and unwavering availability have been instrumental in navigating confusions and queries, ensuring the success of this study's proposal and subsequent writing for the study’s completion.The researcher expresses sincere appreciation to his instructor, Mr. Arwin Rañola, for his steadfast encouragement and reminders to persevere, as well as for serving as a source of motivation that inspires the researcher to strive for excellence. To his adviser, Ms. Neila Paglinawan-Muñez, for her exceptional expertise in guiding the researcher through the intricacies of the study and for her invaluable insights into the analysis techniques employed. Her unwavering availability for consultations, insightful guidance, and constant encouragement have been instrumental in the researcher's pursuit of holistic excellence. The researcher would like to extend his deepest gratitude to Sir Ranzolin Bayeta, PhD, for his invaluable guidance throughout the course of this research. As the Chair of the Panel and the Chair of the Research Department, Dr. Bayeta's insightful feedback, and profound knowledge have been instrumental in the completion of this study. The researcher also extends heartfelt gratitude to his closest friends during his four years in college: Mahalah, Eden, Samuel, Frence, Jane, Rennlyn, and Carl. Their unwavering support and encouragement unique in their own way, have been invaluable throughout this journey. To Mr. Frence Clifford Pillodar, words cannot adequately express the researcher's heartfelt gratitude for generously allowing him to borrow your computer, which was essential for completing his study. From gathering experimental results to writing the manuscript, your assistance was invaluable. Without your help and understanding, the researcher's journey would have been much more challenging. He prays for your success in your studies and aspirations in life. Foremost, the researcher extends heartfelt gratitude to the Almighty God, whose boundless grace and unwavering presence have been the guiding light throughout this journey. In moments of uncertainty and weakness, His strength has been the steadfast anchor, offering solace and fortitude. As the ultimate source of wisdom and enlightenment, His divine counsel has illuminated even the darkest paths, inspiring the researcher to persevere and excel. Truly, He is the greatest friend, mentor, and teacher, enriching every step with His infinite love and blessings.

DEDICATION

I dedicate this book to God, whose endless grace and guidance have been my strength and inspiration. To my family's unwavering support and love. To my sunshine, who brightens my darkest days.

With every day, my longing grows, like the
sun yearns for the sky at
night.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURES **PAGE**

CHAPTER I

INTRODUCTION

Background of the Study

In the real world, tackling complex challenges in image processing, finance, and medicine often involves handling ever-expanding datasets with numerous attributes. These datasets are computationally demanding for machine learning and data mining tasks (Bhattacharyya et al., 2020). Data mining methods, particularly classification and clustering, are crucial for predicting and uncovering insights. However, many datasets contain irrelevant or redundant attributes, complicating the task and degrading the model performance. Identifying and removing these unnecessary attributes before training is essential (Ghosh et al., 2020).

One effective way to address this challenge is through feature selection, which aims to preserve the most important attributes for optimal machine learning performance. Traditional feature selection methods, such as exhaustive, greedy, and random searches, often struggle with complexity, time consumption, and local optima issues (i.e., a feature subset that appears optimal within the current search but may not be the best globally) (Agrawal et al., 2021).

Wrapper methods, which use a learning algorithm to evaluate the performance of feature subsets, are particularly effective for feature selection. These methods tailor the selection process to specific algorithms such as K-Nearest Neighbors (KNN) and Support

Vector Machines (SVM), enhancing the accuracy and efficiency of the resulting models (Agrawal et al., 2021).

Metaheuristic algorithms have gained prominence as viable alternatives for feature selection. These algorithms possess the unique ability to find optimal or near-optimal solutions within a reasonable timeframe (Mirjalili et al., 2014). They are characterized by two key attributes: exploration and exploitation. Exploration allows them to explore the entire solution space, avoiding local optima that can hinder progress. On the other hand, exploitation enables the discovery of better solutions in the vicinity of existing ones, leading to faster convergence (i.e., less time to find an optimal feature subset) (Ghosh et al., 2019).

Recent progress in feature selection has introduced various metaheuristic algorithms, such as the Monarch Butterfly Optimization (MBO) and Mayfly Algorithm (MA). To enhance effectiveness, researchers have developed hybrid algorithms that combine multiple metaheuristics, balancing exploration and exploitation (Bhattacharyya et al., 2020; Eid et al., 2021; Kareem et al., 2022). These hybrids often outperform individual methods, though they require innovative, mathematically sound approaches to effectively tackle large-scale real-world problems (Ting et al., 2015).

Recently, a novel hybrid metaheuristic algorithm for feature selection was proposed by Bhattacharyya et al. (2020). The algorithm combines the Mayfly Algorithm (MA) by Zervoudakis and Tsafarakis (2020) with Harmony Search (HS) by Geem et al. (2001) called Mayfly in Harmony (MA-HS). However, MA-HS has been reported to sometimes suffer from premature convergence and significant execution times in some benchmark datasets.

This paper is driven by the No Free Lunch (NFL) theorem of Wolpert and Macready (1997), which emphasizes that no single metaheuristic algorithm can be universally effective for all types of datasets. Recognizing the diversity in problem characteristics and complexities, we propose a novel hybrid metaheuristic algorithm called Monarch Mayfly Optimization (MMO). This algorithm amalgamates Monarch Butterfly Optimization (MBO) by Wang et al. (2019) and MA, with the aim of achieving a more optimal balance between exploration and exploitation. MMO addresses the limitations of MA-HS, such as slow execution time and premature convergence, and aims to enhance performance in classification tasks.

Statement of the Problem

The purpose of the study is to hybridize MBO and MA to create a novel hybrid algorithm for the feature selection problem, aiming for improved performance and an optimally balanced exploration-exploitation equilibrium. The novel hybrid algorithm, called Monarch Mayfly Optimization (MMO), was evaluated using K-Nearest Neighbors (KNN) and Support Vector Machine (SVM) learning algorithms. Specifically, this study explored the following problems in depth:

1. How does the number of features selected by MMO vary across best fitness and worst fitness when applied to KNN and SVM for a specific dataset?
2. How does the classification accuracy performance of MMO-KNN and MMO-SVM classifiers compare to their standard KNN and SVM counterparts across various datasets based on best fitness and worst fitness?
3. How does the performance of the proposed MMO method compare to the MA-HS method, particularly concerning feature selection and classification accuracy

across various datasets when utilizing KNN and SVM classifiers based on best fitness, worst fitness, and average?

4. What is the time complexity of the proposed MMO?

Objectives of the Study

The study aimed to develop an effective optimization algorithm for feature selection that will combine the behavioral mechanisms employed in MBO and MA to develop a hybrid algorithm. To this end, the study:

1. investigated the variation in the number of features selected by MMO across best and worst fitness scores when employed with KNN and SVM classifiers on a specific dataset;
2. compared the classification accuracy performance of MMO-KNN and MMO-SVM classifiers with their standard KNN and SVM counterparts across diverse datasets, focusing on best fitness and worst fitness scenarios;
3. assessed the performance of the proposed MMO method in comparison to the MA-HS method, specifically regarding feature selection and classification accuracy across multiple datasets when utilizing KNN and SVM classifiers, considering best fitness, worst fitness, and average outcomes; and
4. determined the time complexity of the proposed MMO by performing asymptotic analysis.

Scope and Limitation

The scope of the study was to investigate the hybridization of metaheuristic algorithms for optimization, such as the Monarch Butterfly Optimization (MBO) and

Mayfly Algorithm (MA). The proposed hybrid algorithm aimed to identify and remove redundant and irrelevant features from datasets to improve the performance of machine learning models. Consequently, the study evaluated the performance of the proposed algorithm on various datasets from diverse fields such as biology, chemistry, physics, politics, game theory, and others.

The study was limited to the feature selection side of the feature reduction problem and only covered the classification side of machine learning tasks. The study utilized both binary and multi-class datasets sourced from the University of California, Irvine (UCI) data repository for machine learning. It was crucial to emphasize that all the datasets under consideration exhibited a continuous nature. This signified that the data points within these sets were not discrete or categorical but rather spanned a continuous range, allowing for real-number values, including decimals. While the datasets were diverse, they may not have encompassed all types of real-world datasets due to their limitation to the 18 UCI benchmarking datasets commonly used in research studies like this one. Consequently, the proposed algorithm's generalizability to all real-world datasets remains uncertain.

Significance of the Study

This study introduced a novel hybrid algorithm, called Monarch Mayfly Optimization (MMO), which combined the Monarch Butterfly Optimization (MBO) with Mayfly Algorithm (MA). MMO aimed to explore the in-depth potential of hybridizing MBO wth MA as an approach to the feature selection problem. The findings of this study had the potential to be highly significant and beneficial to a wide range of stakeholders and areas, including, but not limited to:

Machine learning practitioners. By enabling the development of more accurate and efficient machine learning models, the proposed hybrid algorithm has the potential to improve decision-making across a wide range of fields, including healthcare, finance, and manufacturing.

Computer scientists. The proposed hybrid algorithm could help computer scientists to develop more efficient feature selection algorithms. This has the potential to not only make feature selection more accessible to a wider range of users but also lead to the development of new and more powerful machine learning models.

Researchers. The proposed hybrid algorithm could help researchers to better understand the behavior of hybrid metaheuristic algorithms. This could lead to the development of new and more effective hybrid metaheuristic algorithms for a variety of optimization problems.

Computer Science Undergraduate Students. The proposed hybrid algorithm could be used to develop a valuable resource for students who are interested in learning about feature selection and hybrid metaheuristic algorithms. In addition, it could be used as a valuable teaching tool for students interested in machine learning, computer science, or optimization.

Business Firms. Companies that develop or use machine learning models could benefit from the proposed hybrid algorithm by using it to improve the performance of their models.

Society. Society as a whole could benefit from the proposed hybrid algorithm through the development of more accurate and efficient machine learning models that can be used to solve real-world problems.

CHAPTER II

REVIEW OF RELATED LITERATURE

In this chapter, a comprehensive review of related literature will be presented including feature selection, metaheuristic algorithm, and other relevant topics for this study. Additionally, this chapter seeks to establish the context and provide a theoretical framework that propels this study forward.

Feature Selection

Large datasets are commonly encountered in fields such as image processing, finance, business, and medicine. These datasets can create significant computational challenges, particularly for tasks like classification, which involves predicting categories, and clustering, which involves grouping similar data points without labels.

However, not every feature in these datasets enhances the performance of machine learning models. Irrelevant or redundant features can, degrade model performance. To mitigate this problem, feature reduction techniques, including feature selection, are used. These techniques strive to decrease the number of features while preserving or improving the model's accuracy (Bhattacharyya et al., 2020; Ghosh et al., 2020).

Feature selection, in particular, is a vital preprocessing step in machine learning. It involves identifying and choosing a subset of pertinent features from the initial dataset. This selection process helps to improve the model's efficiency and effectiveness by removing irrelevant data (Arora et al., 2019; Liu & Yu, 2005).

Feature selection involves several key steps: acquiring the original dataset, extracting relevant features, establishing evaluation criteria, applying selection rules, and validation (Sharma & Kaur, 2021). These steps can be illustrated using the Breast Cancer Wisconsin (Original) dataset from the University of California, Irvine (UCI) Machine Learning Repository, hereafter referred to as the *Breastcancer* dataset. The *Breastcancer* dataset categorizes cells as benign or malignant tumors (Wolberg, 1992).

Table 1. Breastcancer dataset and its features

Dataset	Features
Breastcancer	Clump thickness
	Uniformity of cell size
	Uniformity of cell shape
	Marginal adhesion
	Single epithelial cell size
	Bare nuclei
	Bland chromatin
	Normal nucleoli
	Mitoses

The original dataset encompasses features providing insights into breast cancer biopsy samples, aiming to discern benign from malignant cells as shown in Table 1.

During feature subset selection, the focus is on identifying pertinent features using a feature selection algorithm. For example, attributes like *Clump Thickness*, *Uniformity of Cell Size*, and *Bare Nuclei* are recognized as crucial for predicting cell malignancy. Once potential features are identified, a learning algorithm evaluates the selected features' performance using metrics including accuracy, precision, recall, and F1-score. These metrics serve as evaluation criteria to assess the model's effectiveness. Selection criteria involve the application of specific rules to optimize the model's performance based on predefined evaluation criteria. This step ensures that the model achieves the best possible outcomes. Validation is then conducted to confirm the model's ability to generalize and maintain accuracy on unseen data. This step ensures the reliability of the selected features in real-world applications. By systematically following these steps, the feature selection process reduces the number of features in the breast cancer dataset, focusing on the most relevant ones. This results in developing a robust model capable of accurately classifying cells as malignant or benign.

From an algorithmic perspective, feature selection poses a difficult binary optimization challenge. The goal is to identify a subset of features from a dataset that best predicts the target variable. This solution is represented by a binary vector, where each element indicates whether a feature is selected (1) or not (0). The algorithm iteratively assesses and refines these solutions to identify the optimal subset (Bhattacharyya et al., 2020).

Despite its seemingly straightforward nature, feature selection becomes more complex with high-dimensional data. As the number of features increases, the number of possible feature subsets grows exponentially, leading to a combinatorial explosion. For example, a dataset with numerous features can generate an enormous number of possible subsets, making exhaustive evaluation computationally infeasible (Guyon & Elisseeff, 2003). This complexity classifies feature selection as an NP-hard combinatorial optimization problem, underlining the computational difficulties involved (Bhattacharyya et al., 2020; Ghosh et al., 2020).

Feature selection approaches vary based on two primary characteristics: search space-based and strategy-based methods. According to Sharma and Kaur (2021), search space-based techniques encompass exhaustive, random, heuristic, and metaheuristic approaches. These methods explore different feature subset spaces, each with its computational implications. While exhaustive methods consider all possible feature subsets, they often become impractical for large datasets due to their computational intensity. Conversely, random methods select subsets randomly, providing computational efficiency but lacking optimality guarantees. Heuristic approaches leverage domain-specific knowledge to guide feature selection, aiming to balance computational efficiency with performance. On the other hand, metaheuristic algorithms adapt and evolve to efficiently navigate the search space, often yielding effective solutions.

In contrast, Jović et al. (2015) classify search techniques into three categories: exponential, sequential, and randomized selection strategies. Exponential strategies systematically increase the number of evaluated features, providing accurate results but can be computationally expensive. Sequential algorithms add or remove features individually, potentially leading to local optima. Randomized algorithms introduce randomness to prevent getting trapped in local optima, employing techniques like simulated annealing and metaheuristics.

The differences between Sharma and Kaur (2021) and Jović et al. (2015) categorizations reflect varying perspectives and levels of detail. While Sharma and Kaur's classification offers a broader view based on overall strategy, Jović et al.'s categorization delve deeper into the nature of search methods. Both classifications provide valuable insights into feature selection approaches.

Filter and wrapper methods also play a significant role in feature selection. Filter methods prioritize feature ranking based on predefined criteria like feature-target variable correlations, operating autonomously without relying on specific learning algorithms. Although they offer faster execution, they may not match the accuracy of wrapper methods. In contrast, wrapper methods employ machine learning algorithms to assess and select feature subsets, integrating closely with the classification process. Although wrapper methods are computationally demanding, they tend to yield more accurate results than filter methods (Agrawal et al., 2021; Sharma & Kaur, 2021).

Metaheuristic Algorithms

This review delves into the versatile nature of metaheuristic algorithms, exploring their stochastic characteristics, adaptability across various domains, and classification based on distinct solution approaches. Metaheuristic algorithms are robust tools for solving optimization problems by identifying optimal or near-optimal solutions from a vast search space without requiring derivative calculations (Mirjalili et al., 2014). Unlike traditional gradient search techniques, these algorithms leverage randomness in solution generation and are known for their simplicity and flexibility, making them adaptable to diverse problem sets. Notably, they mitigate premature convergence by exploring the search space effectively and navigating away from local optima, often likened to a "black box" due to their opaque internal processes. The exploration-exploitation balance is vital, with exploration extensively probing the search space and exploitation focusing on refining promising areas. Metaheuristics find applications across numerous fields, including engineering, data mining, and communication (Olorunda & Engelbrecht, 2008; Mohamed et al., 2020).

Metaheuristic algorithms can be broadly categorized based on solution approaches into single solution-based and population-based algorithms. While single solution-based techniques risk getting trapped in local optima, population-based algorithms evolve multiple solutions over iterations, enabling extensive exploration and collaboration among solutions (Agrawal et al., 2021). Additionally, metaheuristic algorithms are categorized based on behavior, with evolution-based, swarm intelligence-based, physics-based, and human-related algorithms being prominent categories (Mohamed et al., 2020).

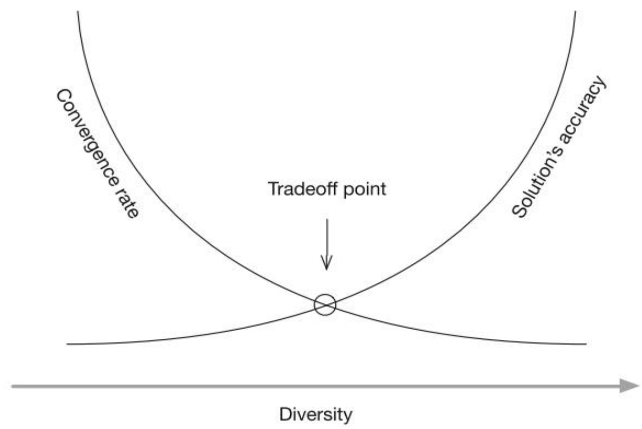

Figure 1. Compromising accuracy and convergence rate (Ting et al., 2015)

Challenges such as premature and slow convergence hinder metaheuristic algorithms in addressing global and highly nonconvex optimization problems. These challenges are intricately linked to solution diversity, as illustrated in Figure 1, with diverse populations fostering exploration and preventing premature convergence. However, achieving diversity poses a tradeoff, as high diversity may lead to slow convergence. The ideal scenario lies at the intersection of rapid convergence and high accuracy, constituting the tradeoff point.

Various hybrid algorithms address these challenges by combining different metaheuristic approaches, aiming to improve convergence rates and exploration diversity (Sheikh et al., 2020; Bhattacharyya et al., 2020; Al-Wajih et al., 2021). Hybrid metaheuristics are further categorized into collaborative hybrids and integrative hybrids. Collaborative hybrids combine multiple algorithms either sequentially or in parallel, facilitating exploration and exploitation across the search space. Integrative hybrids embed subordinate algorithms within master metaheuristics, enhancing solution diversity and convergence rates. These approaches offer tailored solutions to the complex optimization challenges encountered in real-world applications (Ting et al., 2015).

Transfer Functions

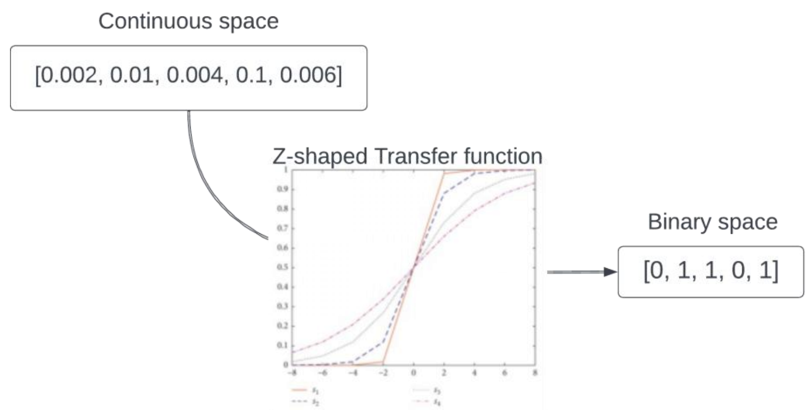

Figure 2. Mapping of continuous space to binary space using a Z-shaped transfer function

Transfer functions serve as essential tool facilitating the transition of solutions from continuous spaces to binary spaces as shown in Figure 2 (Guo et al., 2020). These functions are particularly necessary for metaheuristic algorithms as they help in adapting and applying these algorithms to various optimization problems, including feature selection.

Several transfer functions exist, each with its characteristics and suitability for different optimization tasks. Examples include *S*-shaped functions like sigmoid functions, which generate values in the range [0, 1]. Other types include *V*-shaped and *U*-shaped functions, designed to address specific challenges such as convergence to local optima. For instance, sigmoid functions were initially introduced for binary PSO but may suffer from issues like potential divergence and difficulty in handling local optima (Kennedy & Eberhart, 1997). In contrast, *V*-shaped and *U*-shaped functions aim to overcome these limitations by offering better exploration and exploitation capabilities. However, challenges such as stagnation and parameter tuning persist with these functions (Nezamabadi-pour et al., 2008; Islam et al., 2017; Mirjalili et al., 2020).

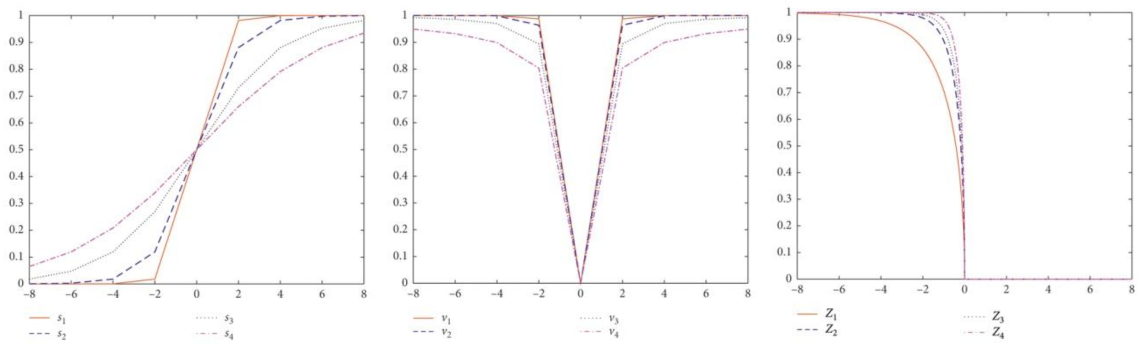

Figure 3. S-shaped, V-shaped, and Z-shaped transfer functions (Guo et al., 2020)

Figure 3 shows the graphical representation of some *S*-shaped, *V*-shaped, and *Z*-shaped transfer functions. Overall, transfer functions serve as indispensable components in metaheuristic algorithms, aiding in the conversion of continuous solutions to binary representations. They enable efficient exploration and exploitation of solution spaces, thus enhancing the algorithm's ability to find optimal solutions in complex optimization problems.

Mayfly Algorithm

The Mayfly Algorithm (MA) was introduced by Zervoudakis and Tsafarakis (2020). MA is a nature-inspired optimization algorithm designed to solve complex optimization problems by mimicking the swarming behavior of mayflies. The algorithm draws inspiration from the life cycle and mating behavior of mayflies, incorporating their natural processes into its optimization strategy. MA seeks to efficiently explore solution spaces and find optimal or near-optimal solutions to challenging optimization problems.

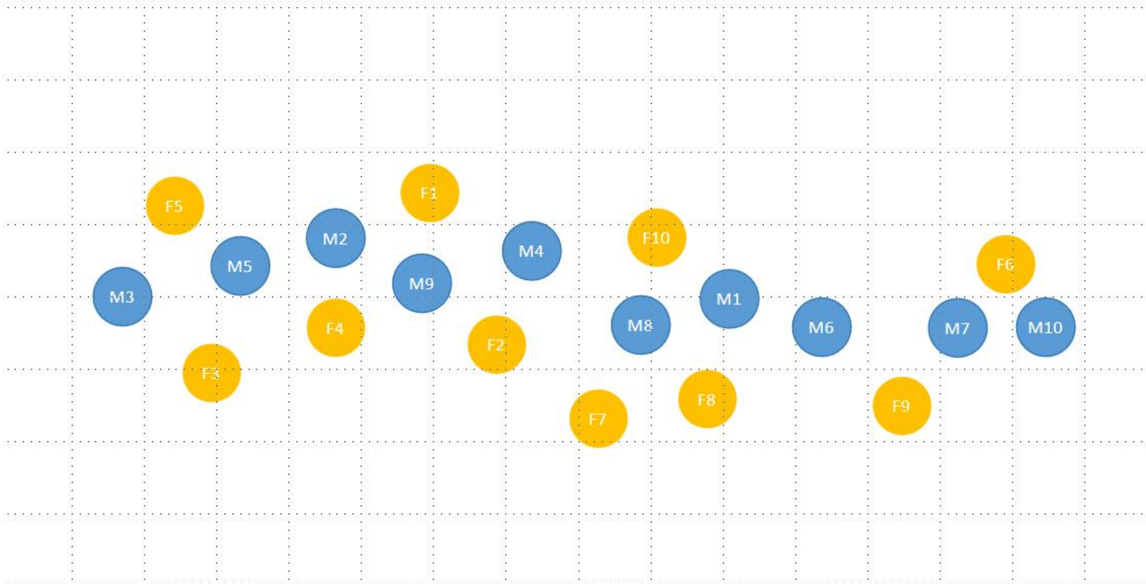

Figure 4. Initial positions of male (blue) and female (yellow) mayflies in the search space

Movement of Male Mayflies

In the initial stage of the Mayfly Algorithm, a population of mayflies is randomly distributed across the search space. Each mayfly represents a potential solution to the optimization problem. The illustration in Figure 4 above shows the initial positions of male and female mayflies, indicated by blue and yellow circles respectively. These positions are analogous to different points in the solution space where the algorithm begins its search for the optimal solution.

Male mayflies update their positions using a formula derived from their velocities, which are influenced by several factors. This update process is governed by using Equation 1 given as

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{1}$$

and Equation 2 given as

$$v_{kj}^{t+1} = g * v_{kj}^t + a_1 * e^{-\beta r_p^2} * \left(pbest_k \quad - x_{kj}^t \right) + a_2 * e^{-\beta r_g^2} * \left(gbest_j \quad - x_{kj}^t \right) \tag{2}$$

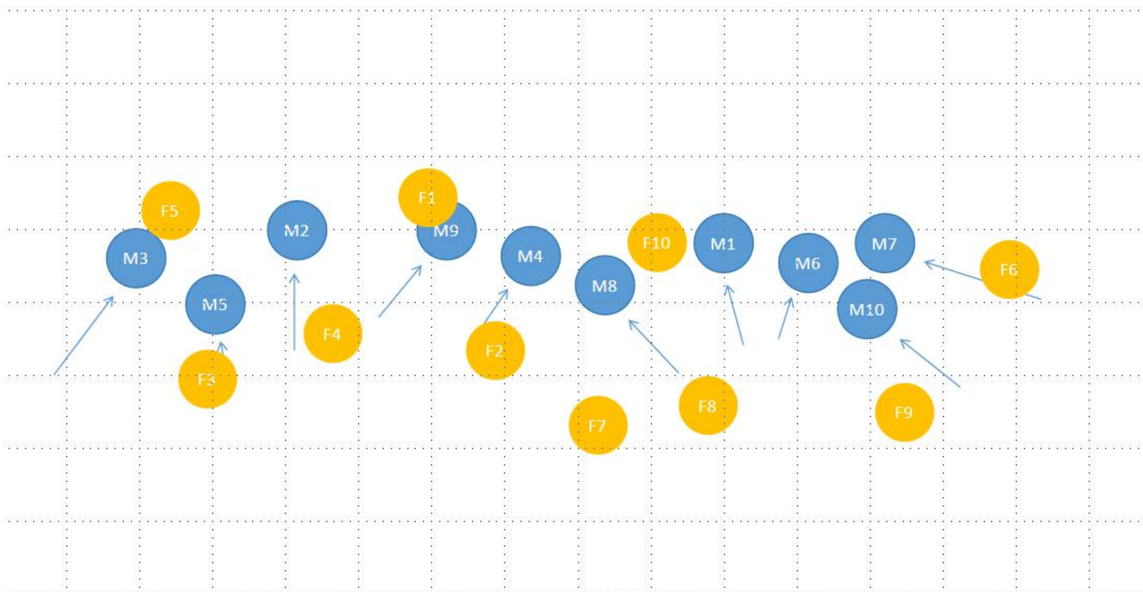

Figure 5. Movement of male mayflies towards their new positions based on their velocity and best known positions

After calculating the new velocities, the male mayflies move towards their new positions based on these velocities and their best-known positions. The illustration of how they move from their previous positions to new ones is shown in Figure 5. To understand how mayflies update their positions, we start with Equation 1, which explains that a male mayfly's position is determined by adding the current position to its new velocity. This shows how the mayflies move through the search space.

Equation 2 describes how to calculate this new velocity v_{kj}^{t+1} for a male mayfly in a particular dimension j. The new velocity is influenced by several factors. A gravitational factor g that affects overall movement. The attraction constants $a_1$and a_2 that measure how much the mayfly is influenced by its own best position $pbest$ and the best position $gbest$ found by the swarm. The exponential factors $e^{-\beta r_p^2}$ and $e^{-\beta r_g^2}$ that limit the influence of the best positions based on distance. Additionally, the terms $pbest_k - x_{kj}^t$ and $gbest_j - x_{kj}^t$ representing the difference between the best positions and the current position help guide the mayfly towards better solutions. Essentially, these equations describe a mayfly's movement as a balance between its own experiences and the collective wisdom of the swarm, adjusting its path to explore the search space effectively.

In a group of mayflies searching for the best positions in the search space, each mayfly keeps track of its personal best position found so far, known as $pbest_k$. As the mayfly moves to a new position at the next time step x_i^{t+1}, it evaluates the quality of this new position using a fitness function. If this new position is better, meaning it has a lower fitness value than the current $pbest_k$, then $pbest_k$ is updated to this new position as shown in Equation 3. Otherwise, $pbest_k$ remains unchanged.

$$pbest_k = \begin{Bmatrix} x_k^{t+1} \\ if\ fitness\left(x_k^{t+1}\right) < fitness(pbest_k) \end{Bmatrix} \tag{3}$$

To decide on their movement, each mayfly calculates two important distances: r_p and r_g. The distance r_p is the Cartesian distance between the mayfly's current position x_k and its personal best position $pbest_k$, while r_g is the distance between the current position and the global best position $gbest$, which is the best position found by any mayfly in the group. These distances are computed using the Cartesian distance formula, which involves summing the squared differences of each coordinate and then taking the square root of this sum. Specifically, for each coordinate in their positions, the distance is calculated using Equation 4 given as

$$|x_k - X_k| = \sqrt{\sum_{j=1}^{n} \left(x_{kj} - X_{kj}\right)^2} \tag{4}$$

where x_{kj} represents the jth coordinate of the kth mayfly's position, and X_{kj} represents the corresponding coordinate of either $pbest_k$ or $gbest$. By continuously updating their personal best positions and considering these distances, mayflies can effectively navigate towards optimal solutions.

Mayflies that have found good solutions need to continue engaging in the nuptial dance given in Equation 5.

$$v_{kj}^{t+1} = g * v_{kj}^{t} + d * r \tag{5}$$

By doing so, they introduce randomness into the algorithm, which is crucial for exploration. In Equation 5, each mayfly decides its next move by combining two factors. First, it considers its past momentum, like continuing a dance step it's already doing. This is represented by $g * v_{kj}^{t}$. Second, it adds randomness $d * r$ to its movement, encouraging it to explore parts of the search space it might not have considered otherwise.

Figure 6. Male mayflies that have found good solutions performing nuptial dance

Figure 6 shows the up and down movement of male mayflies that have found good solutions performing nuptial dance. Even though these mayflies have found promising solutions, there is always a chance that there exist a better solution in the search space. By maintaining the nuptial dance, they ensure that the search remains dynamic and does not get stuck prematurely. In essence, they keep the algorithm flexible and open to discovering potentially superior solutions.

As the algorithm progresses, this randomness gradually fades away given by Equation 6 as

$$d_{itr} = d_0 \times \delta^{itr} \tag{6}$$

This means that the nuptial dance coefficient, d, decreases over time as the algorithm iterates. d_0 represents the starting value of this coefficient. With each iteration itr, the coefficient is multiplied by δ, a random value between 0 and 1. As a result, the influence of randomness gradually diminishes as the algorithm progresses, allowing the mayflies to rely more on learned information as they search for optimal solutions.

Movement of Female Mayflies

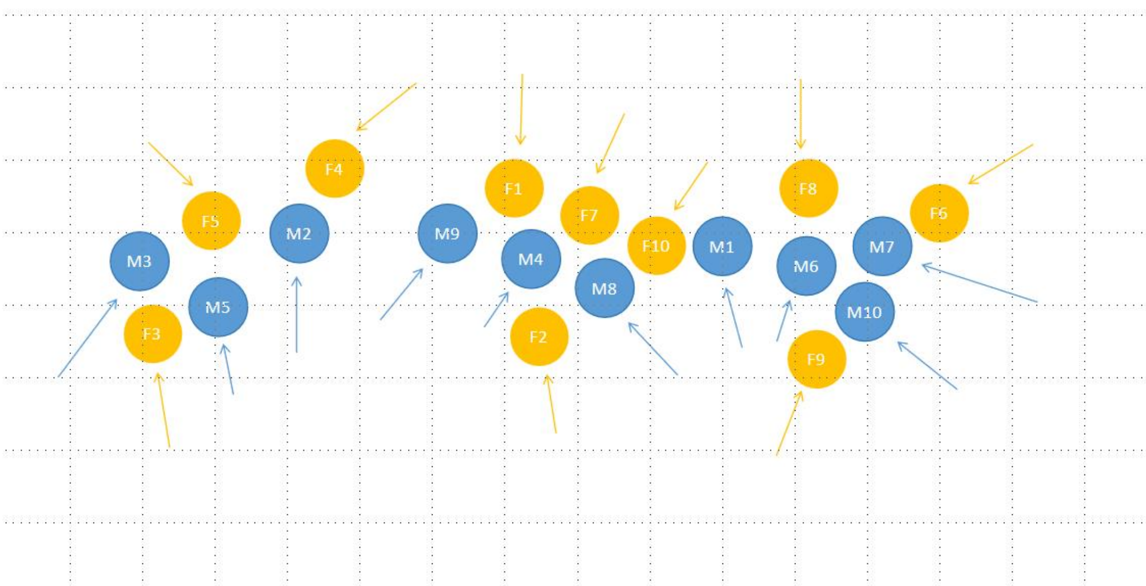

Figure 7. Female mayflies movement towards male mayfies for information exchange

Female mayflies exhibit an intriguing behavior where they actively seek out male mayflies to improve their solutions, as depicted in Figure 7. This behavior mirrors a cooperative effort within the swarm, where females capitalize on the potentially superior solutions of males. Their movement strategy involves updating their position by adding their current position to their new velocity given in Equation 7.

$$y_i^{t+1} = y_i^t + v_i^{t+1} \tag{7}$$

The velocity of a female mayfly v_i^{t+1} is updated based on the quality of the current solution. If the fitness of the female's position y_k is better than the fitness of the male's position x_k, its new velocity is influenced by its previous velocity scaled by a gravitational factor g, and an attraction force towards the male's position. This attraction force is governed by the constants a_2 and β, and the distance between the male and female, r_{mf}. On the other hand, if the fitness of the female's position is not better than the male's, she performs a random walk, influenced by her previous velocity and a random factor. The movement of mayflies as influenced by it's position updates is illustrated in Figure 7. These are represented as in Equation 8.

$$v_{kj}^{t+1} = \begin{Bmatrix} if\ fitness(y_k) > fitmess(x_k) \\ g * v_{kj}^t + a_2 * e^{-\beta r_{mf}^2} * \left(x_{kj}^t - y_{kj}^t\right) \\ else\ if\ fitness(y_k) \leq fitness(x_k) \\ g * v_{kj}^t + fl * r \end{Bmatrix} \tag{8}$$

where fl is a random walk coefficient, and r is a random value between -1 and 1.

The random walk coefficient fl decreases over time according to Equation 9.

$$fl_{itr} = fl_0 \times \delta^{itr} \tag{9}$$

where fl_0 is the initial random walk coefficient, itr is the current iteration, and δ is a random value between 0 and 1. This ensures that the influence of randomness diminishes as the algorithm progresses, allowing female mayflies to effectively explore the search space while being guided by the best solutions found by the males.

Crossover Between Mayflies

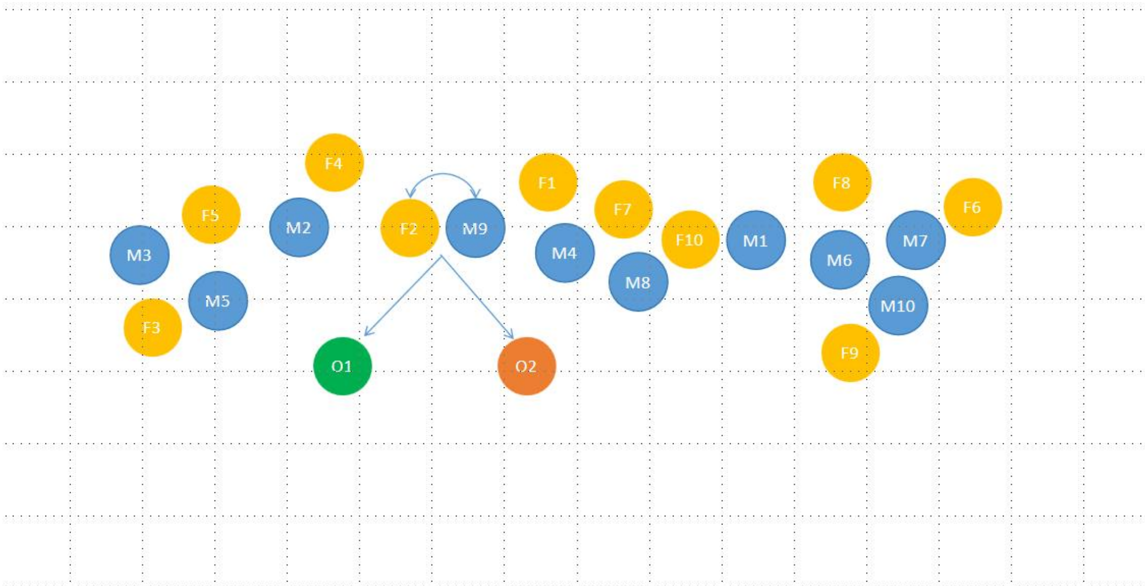

Figure 8. Exchange of genetic information between male and female mayflies to produce two offspring via crossover

The crossover involves exchanging genetic material between male and female mayflies to create offspring shown in Figure 8. The operation begins by selecting the male and female mayflies based on their fitness values, ensuring that the best male mates with the best female. This selection process ensures that the offspring inherit the best traits from both parents, potentially leading to offspring with superior characteristics. This exchange of genetic information promotes diversity within the population, allowing for a broader exploration of potential solutions. Equation 10 and Equation 11 show how the crossover operation produces two offspring.

$$offspring1 = r_{of} * male + \left(1 - r_{of}\right) * female \tag{10}$$

$$offspring2 = r_{of} * female + \left(1 - r_{of}\right) * male \tag{11}$$

In Equation 10, offspring 1 inherits traits predominantly from the male mayfly with a probability of , while in Equation 11, offspring 2 inherits traits predominantly from the female mayfly. This exchange of genetic material between the male and female mayflies promotes diversity within the population and allows for the exploration of different combinations of traits, ultimately enhancing the search for optimal solutions.

Mutation of Mayflies

Mutation helps the algorithm explore new possibilities by introducing small random changes to the offspring. This is done by adding a normally distributed random number k to the offspring's variables, as shown in Equation 12.

$$offspring_n^{'} = offspring_n + k \quad (12)$$

Monarch Butterfly Optimization

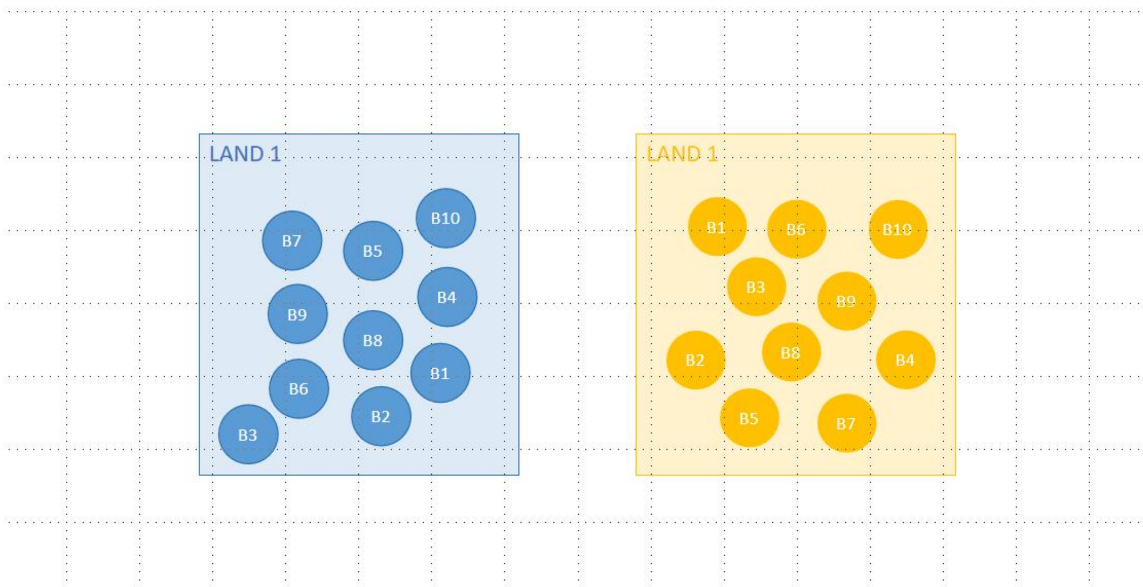

Figure 9. Butterfly population of MBO divided into two lands

The Monarch Butterfly Optimization (MBO) was introduced by Wang et al. (2019). MBO simulates the collective behavior of monarch butterfly populations to solve complex optimization problems. The MBO algorithm divides the butterfly population into two groups, Land 1 and Land 2. Figure 9 illustrates the population of MBO divided into 2 subpopulations. The migration operator governs the movement of butterflies between these two lands, optimizing the overall population. The distribution of butterflies between Land 1 and Land 2 is determined by a ratio parameter p. The total population of butterflies is represented by NP, and the ceiling function is used to round up to the nearest whole number when calculating the number of butterflies in each land. The number of butterflies in Land 1 is calculated as $ceil(p \times NP)$. The number of butterflies in Land 2 is then found by subtracting the number of butterflies in Land 1 from the total population NP.

Migration Operator

Monarch butterflies migrate between Land 1 and Land 2, moving back and forth as part of their natural behavior. During the generation of a new butterfly, the algorithm randomly selects a butterfly from either Land 1 or Land 2 based on a specified ratio.. If a random number r, which is drawn from a uniform distribution, is less than or equal to the ratio parameter p, the algorithm proceeds to generate a new position for the butterfly. This process is guided by Equation 13.

$$x_{i,k}^{t+1} = x_{r_1,k}^{t} \tag{13}$$

In this scenario, the position of the new butterfly is determined by selecting and modifying the position of an existing butterfly from Land 1. Specifically, Equation 13 describes how the new position $x_{i,k}^{t+1}$ of the butterfly i at the next generation $t+1$ is derived based on the position of a randomly chosen butterfly r_1 from Land 1 at the current generation t. This approach ensures that the newly generated butterfly inherits characteristics from the butterflies that currently reside in Land 1, thereby reflecting the influence of this subpopulation in the optimization process.

The random number r is calculated using Equation 15.

$$r = rand * peri \tag{14}$$

Where $peri$ represents the migration period, set to 1.2 (12 months) by Wang et al. (2019). $rand$ is a random number drawn from a uniform distribution.

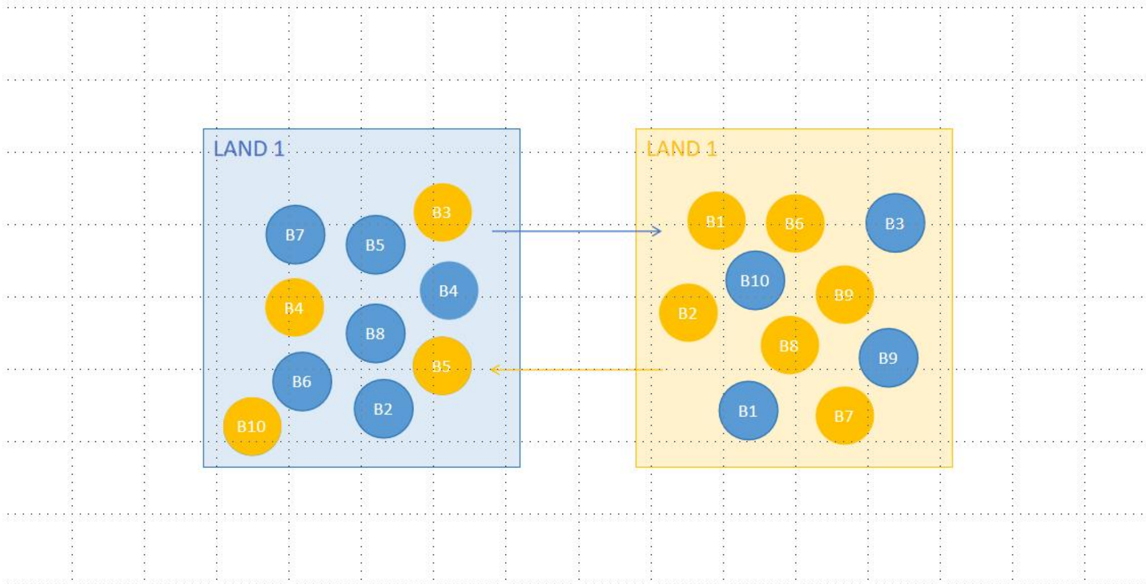

Figure 10. Monarch butterflies migration between to subpopulations

By adjusting the ratio parameter p, the MBO method balances the influence of Land1 and Land 2 on the newly generated butterflies. A larger p means more elements are selected from Land 1, emphasizing its role in the generation process. Conversely, a smaller p shifts the focus to Land 2. Figure 10 illustrates the migration of butterflies between Land 1 and Land 2.

Butterfly Adjusting Operator

In addition to the migration operator, which governs the movement of monarch butterflies between two lands, the Monarch Butterfly Optimization (MBO) algorithm also employs a butterfly adjusting operator to refine the positions of butterflies within their respective subpopulation. The butterfly adjusting operator works through a series of steps. First, for each butterfly j, a random number r is generated. If this number is less than or equal to the probability p, the position of the butterfly is updated to move closeer to the best-known position in the current population as described in Equation 15.

$$x_{j,k}^{t+1} = x_{best,k}^{t} \tag{15}$$

Where $x_{j,k}^{t+1}$ represents the updated position of the kth element of butterfly j in the next generation, and $x_{best,k}^{t}$ denotes the corresponding element of the best butterfly's position in the current generation.

If the random number r is greater than p, the position of butterfly j is influenced by a randomly selected butterfly from Land 2, as specified in Equation 16.

$$x_{j,k}^{t+1} = x_{r_3,k}^{t} \tag{16}$$

This approach introduces diversity into the population by allowing random influences on the butterfly's position. Furthermore, another random number is generated to decide if the position should be adjusted again. If this number exceeds the butterfly adjusting rate (BAR), the position is refined by adding a step influenced by a $Lévy$ flight, as shown in Equation 17.

$$x_{j,k}^{t+1} = x_{j,k}^{t+1} + \alpha \times (dx_k - 0.5) \tag{17}$$

The step size dx is determined by performing a $Lévy$ flight given as in Equation 18, which allows for large jumps in the search space.

$$dx = Lévy\left(x_j^t\right) \tag{18}$$

The extent of this adjustment is controlled by a weighting factor α, defined in Equation 19. The factor α is inversely proportional to the square of the current generation number t, meaning it decreases as the algorithm progresses. A larger α encourages exploration with larger steps, while a smaller α focuses on exploitation with smaller steps. This balance between exploration and exploitation ensures that the algorithm can effectively search the solution space and converge towards optimal solutions.

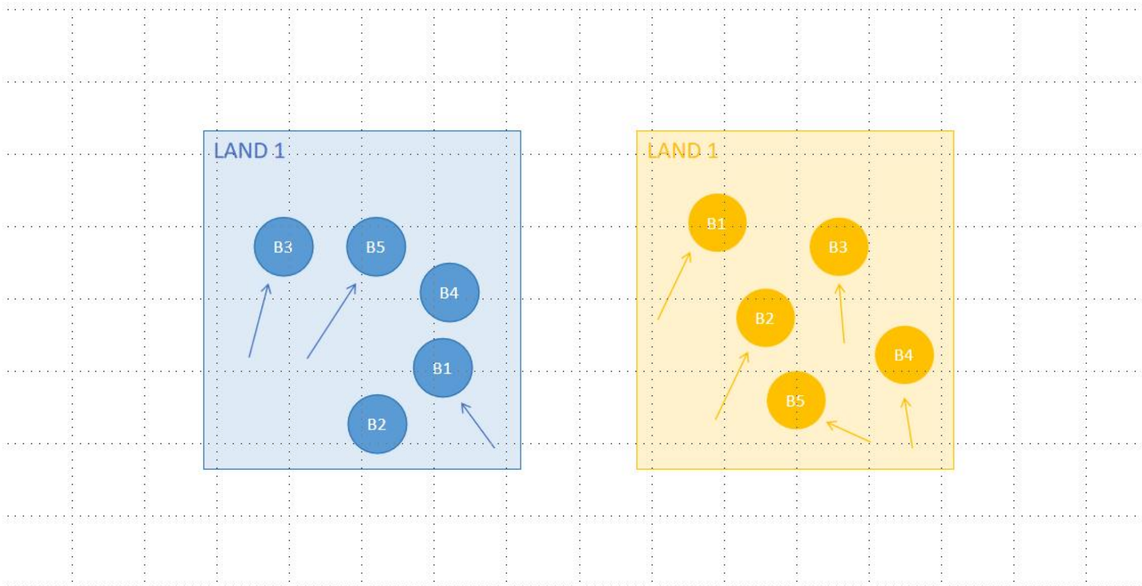

Figure 11. Monarch butterflies position adjustments within their respective lands using adjusting operator

Figure 11 shows how adjusting operator fine-tunes the positions of butterflies within their current lands.

Conceptual Framework

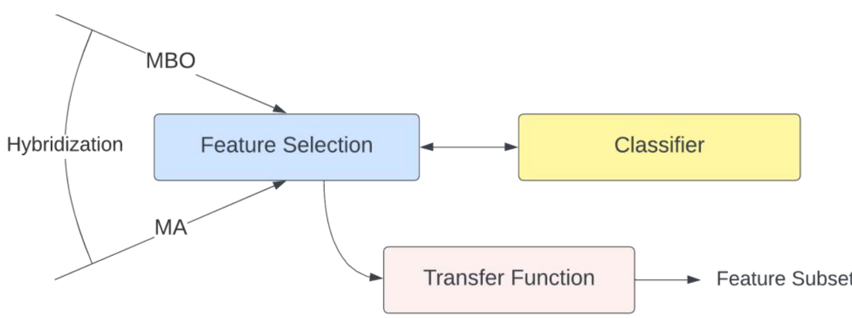

Figure 12. Wrapper-based hybrid metaheuristic algorithm for feature selection problem framework

The conceptual framework outlines a feature selection approach for classification tasks as illustrated in Figure 12. This framework builds upon established practices by integrating feature selection and a machine learning task, in this case, classification. The core of the framework is the wrapper approach, which involves using a learning algorithm to directly evaluate selected features in the context of classification (Sharma & Kaur, 2021).

The feature selection box represents the central component where selected features are evaluated using a learning algorithm. The classifier box is connected to the feature selection box, indicating the learning algorithm's role in feature evaluation. This wrapper approach ensures that the relevance and effectiveness of the chosen feature subset are directly considered in the classification context.

The transfer function box introduces a crucial step by mapping continuous values to binary, essential for adapting the outcomes of metaheuristic algorithms to binary feature subsets. Arrows pointing to the feature selection box signify the application of metaheuristic algorithms (MBO and MA), symbolizing the exploration and optimization of the feature space. The line labeled "hybridization" represents the integration of the Monarch Butterfly Optimization (MBO) and Mayfly Algorithm (MA), combining their strengths to form a new hybrid algorithm called Monarch Mayfly Optimization (MMO). This hybridization enhances the feature selection process, fostering more efficient and effective optimization.

An arrow connecting the feature selection box to the transfer function box denotes the application of a transfer function, ensuring continuous solutions from metaheuristics are adapted to binary feature subsets. The culmination of this process is the generation of a binary feature subset, encapsulating the contributions of the wrapper approach, hybrid metaheuristics, and the transfer function to provide a refined and relevant feature set for improved classification performance.

In summary, this conceptual framework outlines a wrapper approach that incorporates feature selection and classification tasks. The selection of MA and MBO as metaheuristic algorithms is based on their optimization capabilities, while the transfer function aligns continuous solutions to binary outcomes. This approach builds on existing practices, offering a nuanced perspective for refining feature selection processes within classification tasks.

CHAPTER III

RESEARCH DESIGN AND METHODOLOGY

This chapter provides an overview of the adopted approach for the study, covering the research method, materials, equipment, and procedure.

Research Method

In this study, an experimental methodology was employed. Prior to this, it had already been established that metaheuristic algorithms often face challenges such as getting stuck in local optima and exhibiting slow convergence. These issues are primarily due to an imbalanced approach to exploration and exploitation. Achieving a balance between these two processes is difficult, but it is crucial for the success of the algorithms. Diversity plays a significant role in this context. When an algorithm maintains a diverse set of solutions, it can explore the solution space more effectively, which helps in avoiding local optima and enhances the chances of finding a global optimum. Therefore, fostering diversity within the algorithm is essential for achieving an effective balance between exploration and exploitation.

For this reason, a novel hybrid metaheuristic algorithm called Monarch Mayfly Optimization (MMO) was proposed by combining the Monarch Butterfly Optimization (MBO) and Mayfly Algorithm (MA). This hybrid employed a wrapper approach, which involved using a classification or learning algorithm to assess selected features, aligning with the goal of enhancing algorithm performance.

Materials

This study utilized datasets from diverse domains obtained from the University of California, Irvine (UCI) Machine Learning Repository (UC Irvine Machine Learning Repository, n.d.). These datasets covered various fields, such as biology, game theory, chemistry, politics, physics, and more.

Table 2. Benchmark datasets used

#	Dataset	Number of features	Number of samples	Number of classes	Domain
1	Breastcancer	9	699	2	Biology
2	BreastEW	30	569	2	Biology
3	CongressEW	16	434	2	Politics
4	Exactly	13	1000	2	Biology
5	Exactly2	13	1000	2	Biology
6	HeartEW	13	270	2	Biology
7	IonosphereEW	34	351	2	Electromagnetic
8	KrvskpEW	36	3196	2	Game
9	Lymphography	18	148	4	Biology
10	M-of-N	13	1000	2	Biology
11	PenglungEW	325	73	2	Biology
12	SonarEW	60	208	2	Biology
13	SpectEW	22	267	2	Biology
14	Tic-tac-toe	9	958	2	Game
15	Vote	16	300	2	Politics
16	WaveformEW	40	5000	3	Physics
17	WineEW	13	178	3	Chemistry
18	Zoo	16	101	6	Artificial

The datasets and their corresponding information used to test the MMO's performance are shown in Table 2. Additionally, the datasets, all in comma-separated value (CSV) format, encompassed bi-class and multi-class classifications, where instances were categorized into two or multiple classes. It is important to note that all instances in these datasets consisted of continuous values, reflecting the variety and complexity of real-world scenarios. The intention of utilizing datasets from these diverse domains was to evaluate the potential effectiveness and applicability of a method introduced in the course of the research.

Equipment

For this particular experiment, Python was the chosen programming language. The study employed the following libraries: *NumPy* by Harris et al. (2020), *Pandas* by Reback et al. (2020), *scikit-learn* by Pedregosa et al. (2011), and *math* and *random* by Van Rossum (2020). *NumPy* and *Pandas* were essential for the efficient handling and processing of numerical and tabular data, respectively. The *math* and *random* modules supported mathematical computations and for generating pseudo-random numbers, respectively. The *scikit-learn library* provided tools for model evaluation (*accuracy_score*), dataset splitting (*train_test_split*), and the implementation of a K-Nearest Neighbors classifier (KNN) and Support Vector Machines (SVM) classifiers.

Procedure

This section outlines the step-by-step procedures followed in the experiment, starting with the formulation and development of the proposed novel hybrid metaheuristic algorithm for feature selection, Monarch Mayfly Optimization (MMO). Then, it was followed by the subprocesses involved in the algorithm as a whole. Subsequently, an explanation of how the algorithm was evaluated was provided. This included a description of the experimental setup, such as the conditions and parameters. Other relevant details aligned with the objectives of this study were also included.

Monarch Mayfly Optimization

The Monarch Mayfly Optimization (MMO) incorporates a hybrid population structure, drawing inspiration from both the Monarch Butterfly Optimization (MBO) and the Mayfly Algorithm (MA). In the MA, the population is characterized by male and female mayflies, mirroring the natural swarm dynamics observed in mayflies. On the other hand, the MBO introduces a dual-subpopulation model, denoted as $NP1$ and $NP2$, representing subpopulations 1 and 2, respectively, which together constitute the entire monarch butterfly population. NP stands for the total population across both subpopulations. For an in-depth exploration of the population structure of both MBO and MA, a comprehensive description is available in the Monarch Butterfly Optimization and Mayfly Algorithm section in this paper's Chapter II (RRL). Consequently, MMO incorporates two subpopulations and gender-based divisions, which make up the total population. Thus, the population structure of MMO can be expressed mathematically as:

Let S be the swarm size set to some value. Let P_{Sub1} represent the population in subpopulation 1. Let P_{Sub2} represent the population in subpopulation 2. Hence, the total population P_{Total} is the sum of P_{Sub1} and P_{Sub2}, as expressed below in Equation 21.

$$P_{Total} = P_{Sub1} + P_{Sub2} \tag{21}$$

If both subpopulations have a swarm size of S, then $P_{Sub1} = P_{Sub2} = S$. Therefore, the total population size P_{Total} is given in Equation 22 as

$$P_{Total} = 2S \tag{22}$$

$$M_i = R_i \cdot S \tag{23}$$

$$F_i = (1 - R_i) \cdot S \tag{24}$$

In addition, the composition of each subpopulation in terms of males is denoted as M_i and females denoted as F_i can be influenced by a specific ratio denoted as R_i concerning the swarm size S.

Combining both Equation 23 and Equation 24, we get Equation 25

$$M_i + F_i = R_i \cdot S + (1 - R_i) \cdot S \tag{25}$$

which can be simplified in Equation 26 as

$$M_i + F_i = S \tag{26}$$

This expresses the sum of males and females in subpopulation ($i = Sub1,\ Sub2$) as the swarm size S. This mathematical representation encapsulates the idealization of the entire optimization process, providing a clear and flexible framework for understanding the population dynamics within each subpopulation.

INPUT: Positions of male individuals to be updated from the calling subpopulation (migrant_male_swarm_pos),
Positions of female individuals from the calling subpopulation (female_swarm_pos),
Positions of male individuals in the opposite subpopulation (male_swarm_pos),
migration period (peri),
probability threshold (p)

OUTPUT: Updated migrant_male_swarm_pos

Initialize D as all the elements in i^th migrant_male_swarm_pos
  for $i = 1$ to migrant male swarm size **do**
    Evaluate fitness for the current mayfly
    for $k = 2$ to D **do**
      $r = rand * peri$
      if $r \leq p$ **then**
        Randomly select in female swarm
        Generate the k^th element of the x_i^{t+1} as equation (27)
      else
        Randomly select in male swarm
        Generate the k^th element of the x_i^{t+1} as equation (29)
      end if
      Evaluate its fitness
      if new fitness < current fitness
        Update the 0^th element of migrant_male_swarm_pos
      end if
    end for k
  end for i
Return the current migrant_male_swarm_pos

Figure 13. Migration operator

Monarch mayflies remain in their respective subpopulations, but migration between subpopulations occurs through a Migration Operator. Accordingly, the Migration Operator can be represented by its algorithm shown in Figure 13.

The Migration Operator is based on the Migration Operator of the MBO and modified to adapt to the population structure of MMO. The following equations describe the migration process as expressed by Wang et al. (2019), and are discussed in the context of MMO.

$$x_{i,k}^{t+1} = x_{r_1,k}^{t} \tag{27}$$

In the Equation 27, $x_{i,k}^{t+1}$ represents the positions of $NP1$ male mayflies for feature k at the $(t+1)^{th}$ time step. Whereas, $x_{r_1,k}^{t}$ represents the positions of $NP1$ female mayflies for feature k at the t^{th} time step. The equation signifies a migration process where the positions of $NP1$ male mayflies at the next time step $(t+1)$ are replaced by the positions of a randomly selected $NP1$ female mayfly r_1 from the female swarm at the current time step t.

Similarly, for NP2, $x_{i,k}^{t+1}$ represents the positions of $NP2$ male mayflies for feature k at the $(t+1)^{th}$ time step. Whereas, $x_{r_1,k}^{t}$ represents the positions of $NP2$ female mayflies for feature k at the t^{th} time step. The equation indicates that in the migration process for $NP2$ mayflies, the positions of $NP2$ male mayflies at the next time step $(t+1)$ are replaced by the positions of a randomly selected $NP2$ female mayfly r_1 from the female swarm at the current time step t.

The selection of random female mayfly r_1 occurs in line 7 of the algorithm in Figure 13. The Migration continues by replacing the feature k of the male mayfly x_i^{t+1} with the corresponding feature k of the selected female mayfly $x_{r_1}^{t}$ at line 8. However, the condition $r \leq p$ must first be met, where r is a random value scaled by the parameter peri.

Hence, when $r \leq p$, at line 6 of Figure 13, the feature k in the newly generated male mayfly is generated by Equation 27. Here, r can be calculated as

$$r = rand \times peri \tag{28}$$

In Equation 28, $peri$ indicates the migration period and is set to 1.2 (12 months per year) in the work of Wang et al. (2019). The $rand \in [0, 1]$ is a random number drawn from the uniform distribution. In contrast, if $r > p$, the element k in the newly generated mayfly is generated using Equation 29 as

$$x_{i,k}^{t+1} = x_{r_2,k}^{t}. \tag{29}$$

This indicates that the male mayfly position is updated based on the positions of a randomly selected male mayfly r_2. The r_2 can be either a $NP1$ male mayfly or a $NP2$ male mayfly, depending on whether $x_{i,k}^{t+1}$ represents $NP1$ or $NP2$ males. If it is $NP1$ males, r_2 is randomly selected from the $NP2$ male swarm, and vice versa. These occur in lines 10 and 11 of Figure 13.

In summary, the migration process involves updating the positions of male mayflies in $NP1$ and $NP2$ based on the positions of randomly selected female or male mayflies. The decision of selecting a female mayfly or updating based on another male mayfly is probabilistic, determined by the random number $rand$ and the threshold p. This process introduces diversity into the population and facilitates exploration across different regions of the solution space, potentially enhancing the overall performance of MMO.

Aside from Migration Operator, the female mayflies in $NP1$ and $NP2$ can potentially update their positions through perturbation as a mechanism implemented in the Adjusting Operator. This operator is also from MBO modified to adapt in the population structure of MMO. The Adjusting Operator will be discussed in the following paragraphs together with the equations that represent its processes. The following equations describe the Adjusting Operator as expressed by Wang et al. (2019), and are discussed in the context of MMO.

```
INPUT:    Positions of female individuals in female swarm positions (female_swarm_pos),
          Positions of male individuals in the opposite  subpopulation (male_swarm_pos)
OUTPUT:   Updated female_swarm_pos

1  for i = 1 to swarm size do
2    if rand < MAR
3      Randomly generate a number rand by uniform distribution
4      if rand ≤ p
5        if female_swarm_pos ≤ male_swarm_pos
6          Select the x^t_best from female_swarm_pos
7        else
8          Select the x^t_best from male_swarm_pos
9        end if
10       for k = 2 to selected mayfly length do
11         Perform Levy flight perturbation as in Equation (31)
12       end for k
13       Find fitness of selected mayfly
14       Find fitness of i^th female mayfly
15       if selected mayfly fitness ≤ i^th mayfly fitness
16         Generate the i^th element of the x_i^{t+1} by Equation (30)
17       end if
18     else
19       Randomly select a mayfly from the male swarm
20       Find fitness of selected mayfly
21       Find fitness of i^th female mayfly
22       if selected mayfly fitness ≤ i^th female mayfly fitness
23         Generate the i^th element of the x_i^{t+1} by Equation (34)
24       end if
25     end if
26   end if
27 end for i
28 Return the current female_swarm_pos
```

Figure 14. Adjusting operator

The Adjusting Operator is shown in Figure 14. Before the Adjusting Operator is executed, a random number is first generated in range between 0 and 1 at line 2 of Figure 14. When the randomly generated number is less than the mayfly adjusting rate, denoted as MAR, then the execution of adjusting operator will commence. For all elements in female mayfly i, for the second time, a number $rand \in [0, 1]$ is generated. If a randomly generated number $rand$ is smaller than p, where p is a probability threshold, the female mayfly i position can be updated as in Equation 30

$$x_{i,k}^{t+1} = x_{best,k}^{t} \tag{30}$$

where $x_{j,k}^{t+1}$ indicates the kth element of x_j at generation $t+1$ that presents the position of the female mayfly i. Similarly, $x_{best,k}^{t}$ indicates the kth element of x_{best} that is the best mayfly in NP1 and NP2. t is current generation number. This is because, the best mayfly from the male population and female population is considered for diversity.

Since there are two subpopulations $NP1$and $NP2$, the Adjusting Operator is called by $NP1$ female mayflies and $NP2$ female mayflies. When the Adjusting Operator is called in $NP1$, the $NP1$ female and $NP2$ male positions are passed as arguments. When x_{best}^{t} in $NP1$ females has better fitness value than the x_{best}^{t} in $NP2$ males, then it is selected to replace the female mayfly i. Otherwise, the x_{best}^{t} in $NP2$ males is selected, and vice versa when Adjusting Operator is called in $NP2$. This selectiion occurs at line 5 to 8 of Figure 14.

For each k feature element in the selected mayfly x_{best}^{t}, a $Lévy$ flight perturbation is applied as

$$x_{best}^{t} = x_{best}^{t} + \alpha \times (dx_k - 0.5) \tag{31}$$

In Equation 31, dx is the walk step of the selected mayfly i that can be calculated by performing $Lévy$ flight as shown in Equation 32

$$dx = Lévy(size) = \sum_{i=1}^{s} tan(\pi \cdot rand_i) \tag{32}$$

The function $Lévy(size)$ generates a $Lévy$ flight perturbation by summing the tangent of the product of π and randomly generated numbers $rand_i$ for each element from 1 to the specified control parameter step size s. The $Lévy$ flight is influenced by the weight factor α that adaptively adjusts in each iteration as shown in Equation 33

$$\alpha = S_{max}/t^2 \tag{33}$$

where S_{max} is max walk step that the selected mayfly individual can move in one step, at time step t. A larger α corresponds to a longer step in the search space, which increases the influence of dx on $x_{i,k}^{t+1}$ and encourages the process of exploration. This encourages the algorithm to explore new regions more extensively. On the other hand, a smaller α

corresponds to a shorter step, which decreases the influence of dx on $x_{i,k}^{t+1}$ and encourages the process of exploitation. This discourages extensive exploration and encourages the algorithm to exploit known regions more intensively.

After the $Lévy$ perturbation at line 11 of Figure 14, if the fitness of the selected mayfly $x_{best,k}^{t}$ is better than the current female mayfly $x_{i,k}^{t+1}$, then Equation 30 is processed. However, when the randomly generated number $rand$ is greater than the probability threshold p, a random male mayfly is selected from either $NP1$and $NP2$ male maflies depending on which subpopulation the adjusting operator is called. This occurs in line 20 of Figure 14. Hence, Equation 34 takes place:

$$x_{i,k}^{t+1} = x_{r_3,k}^{t} \tag{34}$$

where $x_{r_3,k}^{t}$ indicates the kth element of x_{r_3} that is randomly selected in male mayflies. The fitness of the selected male mayfly r_3 is further evaluated to replace the female mayfly x_i^{t+1} if it has a better fitness value.

Overall, this Adjusting Operator introduces a stochastic element by considering a random $Lévy$ flight perturbation for selected mayflies and, based on fitness comparisons, either updating the female swarm positions with a perturbed mayfly or replacing a female mayfly with a better-fitness male mayfly. The selection of the best mayfly is influenced by their fitness values, and the process is subject to the probabilities specified by the adjusting rate and the random values drawn from uniform and $Lévy$ flight distributions. This can help in exploring the solution space more widely, potentially discovering better solutions in the vicinity of the current positions.

```
INPUT    Position of male individuals in the population swarm (population_male_swarm_pos),
         Velocity of male individuals in the population swarm (population_male_swarm_vel),
         Position of female individuals in the population swarm (population_female_swarm_pos),
         Velocity of female individuals in the population swarm (population_female_swarm_vel),
         Position of elite individuals (elite_swarm_pos),
         Fitness of elite individuals (elite_swarm_fitness),
         Number of elite individuals to select (num_elite),
         None (The function operates by modifying the input parameters directly)
OUTPUT:
if fitness of the best male individual < fitness of the best elite individual:
    Update elite swarm fitness with fitness of the best male individual
    Copy position of the best male individual to elite swarm position
end if
if fitness of the best female individual < fitness of the best elite individual:
    Update elite swarm fitness with fitness of the best female individual
    Copy position of the best female individual to elite swarm position
end if
for each elite individual:
    Replace worst male individual with elite individual's position
    Set velocity of replaced male individual to zero
    Replace worst female individual with elite individual's position
    Set velocity of replaced female individual to zero
end for
```

Figure 15. Elitism

Elitism is introduced to MMO that serves as a mechanism to ensure the preservation and propagation of the best-performing solutions across generations. Elitism functionality begins by comparing the fitness of the best male individual and the best elite individual as shown in Figure 15. If the male individual's fitness surpasses that of the elite, the elite's fitness and position are updated to match those of the male individual. A similar comparison is conducted between the fitness of the best female individual and the best elite individual. If the female individual's fitness exceeds that of the elite, the elite's fitness and position are updated accordingly.

Subsequently, for each elite individual, the position of the worst male individual is replaced with the elite individual's position. Then velocity of the replaced male individual is set to zero. Then the position of the worst female individual is replaced with the elite individuals position. In addition, the velocity of the replaced individual is set to zero.

Another operation used in MMO is the Crossover and Mutation. The Crossover operation entails the exchange of genetic material between male and female mayflies to

```
INPUT:    Position of male individuals in the population swarm (NP_male_swarm_pos),
          Position of female individuals in the population swarm (NP_female_swarm_pos),
          Velocity of male individuals in the population swarm (NP_male_swarm_vel),
          Velocity of female individuals in the population swarm (NP_female_swarm_vel),
          Total number of features. subpop_size: Size of the subpopulation (tot_features),
OUTPUT:   Crossover random value r_of (l)
          Updated position of male individuals in the population swarm (NP_male_swarm_pos),
          Updated position of female individuals in the population swarm (NP_female_swarm_pos),
          Updated velocity of male individuals in the population swarm (NP_male_swarm_vel).
          Updated velocity of female individuals in the population swarm (NP_female_swarm_vel)

NP_offspring1 = array of zeros with dimensions (subpop_size, tot_features)
NP_offspring2 = array of zeros with dimensions (subpop_size, tot_features)
for each individual in the subpopulation:
    Generate a random partition point within the feature space
    for each feature:
        Calculate the value of the feature for offspring 1 using crossover
        Calculate the value of the feature for offspring 2 using crossover
    end for
    Randomly select one of the offspring for male individual and the other for female individual
    for each feature:
        Introduce mutation for the male and female individuals
    end for
    Reset velocities for male and female individuals
end for
Return NP_male_swarm_pos, NP_female_swarm_pos, NP_male_swarm_vel, NP_female_swarm_vel
```

Figure 16. Crossover and mutation

generate offspring as depicted in Figure 16. Through a selection process based on their fitness values, the algorithm ensures that the offspring inherit advantageous traits from both parents. This genetic exchange fosters diversity within the population and facilitates exploration of various trait combinations, potentially yielding offspring with superior attributes. Additionally, Mutation introduces minor random alterations to the offspring by incorporating a normally distributed random number into their variables. This stochastic perturbation aids the algorithm in exploring novel avenues and seeking optimal solutions by injecting diversity into the population.

The subsequent paragraphs will delve into the core MMO process, with a focus on the novel behaviors introduced within the overall optimization framework. While the migration, adjusting operators, and elitism of MBO are already integrated, the discussion will highlight the aspects of the algorithm that introduce fresh dynamics to the optimization process.

```
INPUT:      Subpopulation 1 male and female swarm (NP1),
            Subpopulation 2 male and female swarm (NP2),
            Maximum  iteration (MaxIter)
OUTPUT:     Best solution vector Agent X = [x1, x2, …, xd] found by MMO, which represent the selected features

Initialize NP1, NP2 population and velocity of male and female  mayflies randomly
Evaluate population and then find gbest
for itr = 1 to MaxIter do
    Update velocity limit for NP1 and NP2 swarm
    for i = 1 to NP1
        Update gbest
        Update pbest
        Evaluate and update the velocities of NP1 male and femalemayflies
    end for
    Levy flights for NP1 female mayflies
    Perform migration operator on NP1 males
    Perform adjusting operator on NP1 females
    Calculate fitness of NP1 male and female mayflies as illustrated in Figure 18
    Sort NP1 male and female mayflies and rank them according to fitness scores
    Perform crossover on NP1 mayflies and generate male and female offspring and mutate the offspring
    Replace worst NP1 mayflies with the best new offspring generated
    Update position for NP1 males and females swarm
    Apply Elitism on NP1 maylies
    Update gravity and nuptial dance
    Update gbest
    while counter < MaxIter
        for i = 1 to NP2
            Update gbest
            Update pbest
            Evaluate and update the velocities of NP2 male and female mayflies
        end for
        Levy flight for NP2 female mayflies
        Perform migration operator on NP2 males
        Perform adjusting operator on NP2 females
        Calculate fitness of NP2 male and female mayflies as illustrated in Figure 18
        Sort NP2 male and female mayflies and rank them according to fitness scores
        Perform crossover on NP2 mayflies to generate male and female offspring and mutate the offspring
        if new gbest < current gbest
            Set current gbest to new gbest
            Reset counter when a better solution is found
        else
            Increment the counter when no improvement is found
        end if
        Perform crossover on NP2 mayflies and generate male and female offspring and mutate the offspring
        Replace worst NP2 mayflies with the best new offspring generated
        Update position for NP2 males and females swarm
        Apply Elitism on NP2 maylies
        Update gravity and nuptial dance
        Update gbest
    end while
end for
```

Figure 17. Main MMO process

The main MMO process is shown in Figure 17. The Mayfly Optimization (MMO) process begins with the initialization of two populations of mayflies, $NP1$ and $NP2$, each comprising male and female individuals with randomly assigned positions and velocities. The fitness of each mayfly is evaluated, and the global best position ($gbest$) is identified based on the highest fitness score. The main optimization loop runs for a maximum number of iterations ($MaxIter$). Within this loop, the velocity limits for both $NP1$ and $NP2$ swarms are updated. For each mayfly in $NP1$, the global best ($gbest$) and personal best ($pbest$) positions are updated, and the velocities of male and female mayflies are recalculated. Special operations are performed, including $Lévy$ flights for female mayflies to introduce randomness and exploration, Migration Operator for male mayflies, and Adjusting Operator for female mayflies. The fitness of the $NP1$ mayflies is calculated, and they are sorted and ranked accordingly. A Crossover operation is performed to generate offspring, followed by Mutation. The worst-performing mayflies are replaced with the best new offspring, and the positions of $NP1$ mayflies are updated. Elitism is applied to retain the best individuals, and parameters related to gravity and nuptial dance behaviors are adjusted. The global best position is updated once more.

Simultaneously, a similar process occurs for the $NP2$ population. Each mayfly in $NP2$ undergoes velocity updates, and operations such as $Lévy$ flights, Migration, and Adjusting are performed. Fitness is recalculated, and mayflies are ranked. Crossover and Mutation generate new offspring, replacing the worst performers with the best new individuals. Positions are updated, Elitism is applied, and gravity and nuptial dance parameters are adjusted. If a new global best is found, the current global best is updated, and the iteration counter is reset; otherwise, the counter is incremented. This loop continues until the maximum number of iterations is reached. The entire process ensures continuous improvement in the fitness of the mayfly populations through iterative optimization.

Since MMO is a wrapper, the calculation of fitness scores is done with the interaction of the proposed MMO to the learning model or classfier. This is illustrated

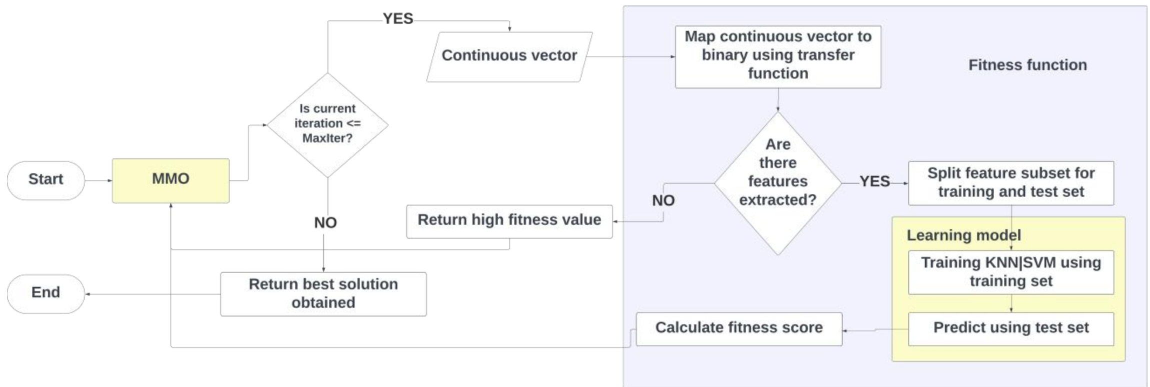

Figure 18. Flowchart of MMO interaction with the classifier through fitness function

as shown in Figure 18. The process begins by generating the initial population of solutions. Once the initial population is generated, the iteration count is initialized. The system then checks whether the iteration count is less than the max iteration ($MaxIter$). If the iteration count is indeed less than $MaxIter$, a solution is selected from the population. This selected solution is then sent to the fitness function. Upon receiving the solution, the fitness function applies a transfer function to it.

The resulting output from the transfer function is examined to determine the selected features. If no features are selected, the fitness function returns a high fitness score as penalty. Which is considered undesirable. After returning this score, the process increments the iteration count and checks again if it is still less than $MaxIter$, thus creating a loop. If the transfer function selects valid features, the fitness function proceeds to train a classifier using these features.

Once the classifier is trained, it is tested on the selected features. Based on the test results, the fitness score is calculated. This calculated fitness score is then returned to the MMO. The MMO uses the returned fitness scores to update the solutions in the population. Following the update of solutions, the iteration count is incremented. The process continues to loop back to check if the iteration count is less than $MaxIter$, iterating through the steps of selection, fitness evaluation, and updating until the iteration count reaches $MaxIter$. At this point, the process comes to an end.

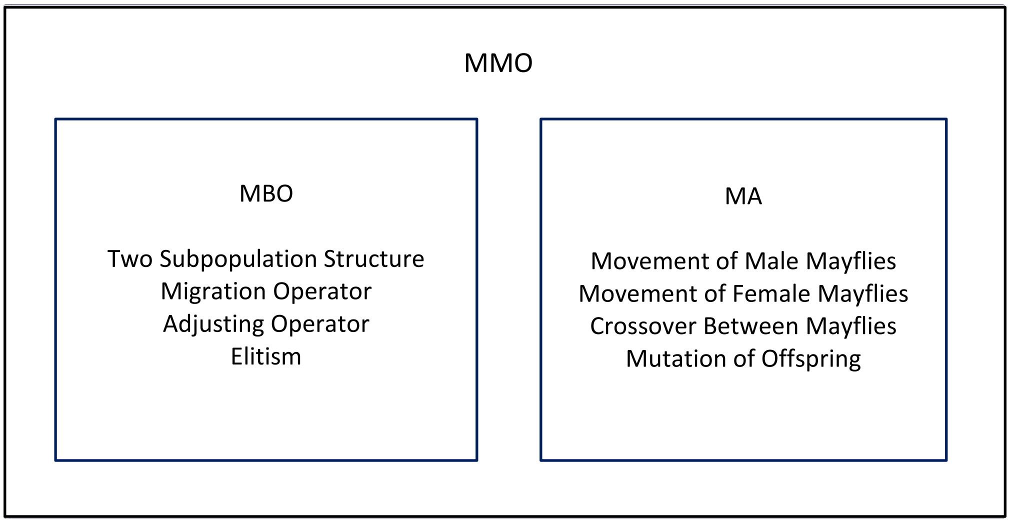

Figure 19. Critical components of MBO and MA used in the proposed hybrid MMO

In a nutshell, the proposed hybrid MMO for the feature selection problem amalgamates various key components to form a cohesive approach as shown in Figure 19. It incorporates MBO's two Subpopulation Structure, comprising a Migration Operator facilitating inter-subpopulation exchange to bolster diversity. An Adjusting Operator refining individuals within subpopulations for localized enhancements. As well as the Elitism to preserve top solutions across generations. Additionally, drawing from the MA, it integrates operations mimicking the behaviors of male and female mayflies. This includes the Movement of Male Mayflies, emulating exploration and exploitation strategies. The Movement of Female Mayflies, guiding the search towards superior solutions. Crossover Between Mayflies combines attributes from parent mayflies to generate diverse offspring, while Mutation of Offspring introduces variability, preventing premature convergence and promoting thorough exploration of the solution space. Together, these components synergize to create a hybrid metaheuristic algorithm capable of effectively tackling the feature selection challenge.

Since each solution vector obtained by the MMO comprises continuous values, it is not directly applicable to address FS problem. Employing a mapping function is imperative to convert these continuous values into binary 0s and 1s. The transfer function play a crucial role in specifying the rate of change in the decision variable values, transitioning them between 0 and 1.This conversion will be facilitated by the utilization of the *S*-shaped transfer function.

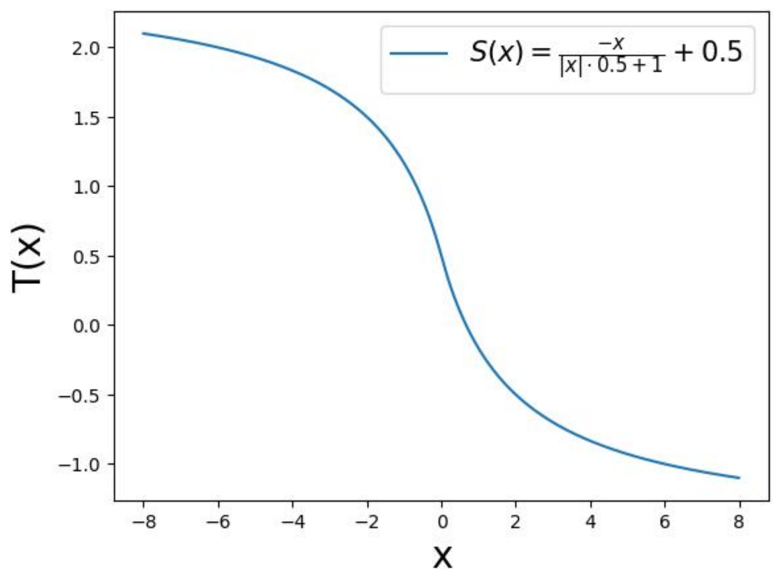

Figure 20. S-shaped transfer function for converting continuous search space into binary

$$S(x) = \frac{-x}{|x| \cdot 0.5 + 1} + 0.5 \tag{35}$$

The S-shaped transfer function given in Equation 35, determines the probability of selecting a specific feature within a solution vector. A graphical representation of the *S*-shaped transfer function, is provided in Figure 20. During the conversion process, the agent's feature is updated according to Equation 36 (Guo et al., 2020).

$$x_{i,j}^{t+1} = \begin{cases} 1 \; if \; S\left(x_{i,j}^{t+1}\right) \geq rand \\ 0 \; if \; S\left(x_{i,j}^{t+1}\right) < rand \end{cases} \tag{36}$$

In which $x_{i,j}^{t+1}$ represents the j^{th} dimension of the i^{th} individual at the current iteration $t + 1$, $rand$ is a number selected randomly from within the range $[0, 1]$, and $S\left(x_{i,j}^{t+1}\right)$ is the probability value obtained when applying a given transfer function to every j^{th} component's continuous value of agent i. Thus, This transfer function maps the continuous values to probabilities, which are then used in the decision-making process.

Before this point, it has been established that the primary goal of FS is to select the least possible number of features while maintaining accuracy at its peak, in this case, for a classification task. The solution vector (feature subset) is assessed using a learning algorithm within a wrapper-based method to get the classification accuracy. Consequently, the fitness function is formulated to encompass both the classification error and the count of selected features, aligning with the overarching objective of achieving a balance between feature reduction and classification accuracy. For this purpose, the fitness function for evaluating the feature subset is given in Equation 37 (Bhattacharya et al., 2020)

$$\downarrow Fitness = \gamma \times \frac{|f|}{|F|} + (1 - \gamma) \times \lambda \tag{37}$$

The term $\gamma \times \frac{|f|}{|F|}$ represents the contribution to the fitness function from the desire to minimize the number of features. Where, $\gamma \in [0, 1]$ is a weight parameter that balances the trade-off between minimizing the number of features and minimizing the classification error. A higher value of γ (near 1) means the algorithm places more importance on simplicity and reducing dimensionality, potentially at the expense of a slight decrease in classification accuracy. On the other hand, a lower value of γ (near 0), means the algorithm prioritizes maintaining or improving accuracy, potentially allowing for a larger number of features in the subset if it contributes to better predictive performance. $|f|$ is the number of features in the feature subset, and $|F|$ is the number of features in the given dataset. Simultaneously, the term $(1-\gamma)\times\lambda$ represents the contribution to the fitness function from the goal of minimizing the classification error. The $(1-\gamma)$ part ensures that the weight assigned to the classification error is complementary to the weight assigned to the number of features. As γ increases, the weight o the classification error decreases, and vice versa. This allows for a flexible adjustment between the two objectives in the fitness function. The chosen classifiers for this evaluation is the k-nearest neighbor (KNN) classifier by Altman (1992) and support vector machines (SVM) by Cortes and Vapnik (1995).

Assessment Metrics

In this study, the first objective was to assess the performance of the proposed new hybrid algorithm MMO, in its generated feature subset. The metrics such as best fitness and worst fitness, were used to provide a comprehensive view of the MMO's behavior and performance across multiple datasets.

The best fitness metric indicates the highest level of quality or performance that the algorithm can attain. This metric helps identify the best solutions or feature subsets that lead to optimal results according to the defined fitness function in Equation 37. This metric refers to the minimum value of the fitness function (Al-Wajih et al., 2021). When the algorithm is run M times, its best fitness is calculated as in Equation 38:

$$Best\ fitness = Min_{k=1}^{M} g_*^k \tag{38}$$

where g_*^k is the optimal fitness value achieved at run k.

The worst fitness helps assess the algorithm's ability to avoid poor solutions, as a lower worst fitness suggests better robustness. This metric refers to the maximum value of the fitness function (Al-Wajih et al., 2021). When the algorithm is run M times, its worst fitness is calculated as in Equation 39:

$$Worst\ fitness = Max_{k=1}^{M} g_*^k \tag{39}$$

where g_*^k is the optimal fitness value achieved at run k.

The second objective of the study aimed to assess how the classifiers' performance differs before and after applying the feature selection algorithm MMO. We compared the average classification accuracy of standard KNN and SVM to their MMO-KNN and MMO-SVM counterparts. That is, the achieved classicifation accuracy of MMO-KNN based on the best fitness and worst fitness when the algorithm is run M times is compared to the average classification accuracy of the standard KNN. The same is done with the comparison between MMO-SVM and its standard counterpart SVM.

When the algorithm is run M times, its average accuracy is calculated as in Equation 40:

$$Average\ accuracy = \frac{1}{M} \sum_{k=1}^{M} Accuracy^k \tag{40}$$

where $Accuracy^k$ is the accuracy achieved at run k. $Accuracy$ computes the ratio of correctly classified instances to the total number of instances as shown in Equation 41.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \tag{41}$$

To achieve objective three, the study compared the proposed method MMO with the recent hybrid method named MA-HS (Bhattacharyya et al., 2020). MA-HS was compared to 12 state-of-the-art (SOTA) wrapper-based metaheuristic feature selection algorithms and was found to be superior overall in optimizing classifier performance. The number of features selected by MMO-KNN and MA-HS-KNN is compared, as well as the number of features selected by MMO-SVM and MA-HS-SVM. The feature selection comparison is based on the features obtained with the best fitness, worst fitness, and average fitness when the algorithms are run multiple times.

The average selected feature size provides a normalized measure by expressing the average size of selected features relative to the total number of features (Al-Wajih et al., 2021). When the algorithm is run M times, its average selection is calculated as in Equation 42:

$$Average\ selection = \frac{1}{M}\sum_{k=1}^{M}\frac{Avg.size^k}{T_f} \tag{42}$$

where T_f is the total number of features, $Avg.size^k$ is the average size of selected features achieved at run k.

Subsequently, the classification accuracy achieved by MMO-KNN is compared with MA-HS-KNN, and similarly for MMO-SVM and MA-HS-SVM. The comparison of the classification accuracy is based on the best fitness, worst fitness, and average obtained when the algorithms are run multiple times.

For objective four, an asymptotic analysis is conducted to determine the time complexity of MMO, focusing on worst-case performance using Big-O notation to represent the upper bound on the algorithm's growth rate (Bimurat et al., 2023). The formal definition of the Big-Oh notation is as follows.

“A function $f(n)$ is said to belong to the class $O(g(n))$, denoted as $f(n)\epsilon O(g(n))$, if $f(n)$ is bounded above by a constant multiple of $g(n)$ for sufficiently large values if n. In other words, there exists a positive constant c and a non-negative integer n_0 such that $f(n) \leq cg(n)$ holds for all $n \geq n_0$”.

CHAPTER IV

RESULTS AND DISCUSSIONS

This chapter deals with the results which support the effectiveness of the Monarch Mayfly Optmization (MMO) for solving the Feature Selection (FS) problem.

Tuning of Parameters

Table 3. Hyperparameters and their corresponding value used in the proposed MMO algorithm

Hyperparameter	Description	Value
S	Swarm size of $NP1$ and $NP2$	20
R_i	Ratio with respect to S	0.5
$MaxIter$	Maximum number of iterations	20
γ	Relative weightage used for fitness value	0.01
$a1$	Positive attraction constant	3
$a2$	Positive attraction constant	3.5
β	Visibility coefficient	0.1
d_0	Initial nuptial dance coefficient	3
fl_0	Initial random walk coefficient	3
r_{of}	Random value for crossover	0.95
g	Gravitational coefficient	0.98
δ	d_0 and fl_0 update multiplier	0.9
s	Levy flight size	1
MAR	Mayfly adjusting rate	0.1
p	Probability threshold for Adjusting Operator	float(5/12)
S_{max}	Max walk-step of mayfly in $Levy$ flight	0.02
$peri$	Migration period	1.2
$MaxNP2Iter$	Iteration limit should NP2 continue without finding a better solution	20
mutation_strength	Mutation strength	1.2

The control parameters of the proposed MMO are presented in Table 3. Each hyperparameter is described along with its corresponding optimized value. The control parameter R_i is set to 0.5. To make male and female mayflies equally for both subpopulations *NP1* and *NP2*. Currently, no experiments are done regarding the effect of the ratio of male and female mayflies for each subpopulation $NP1$ and $NP2$.

Performance Analysis of MMO

In this section, the performance of the proposed feature selection method MMO is evaluated when used as a wrapper with the KNN and SVM classifiers. The evaluation includes an analysis of how the number of features selected varies across fitness scores for all 18 UCI benchmarking datasets. Additionally, the analysis examines how the number of features selected, or the reduction in features, impacts the performance of these classifiers. This analysis aims to determine the effectiveness of the proposed MMO in optimizing feature subsets and improving classification accuracy.

Table 4. MMO-KNN's selected features count based on best and worst fitness scores over 10 runs

MMO-KNN					
Dataset	Original no. of features	Best fitness score	Best Feature Count	Worst Fitness Score	Worst Feature Count
Breastcancer	9	0.0040	4	0.0384	3
BreastEW	30	0.0100	4	0.0621	4
CongressEW	16	0.0025	4	0.0594	4
Exactly	13	0.0046	6	0.0252	7
Exactly2	13	0.1912	4	0.2190	8
HeartEW	13	0.0764	4	0.1696	6
Ionosphere	34	0.0006	2	0.0589	8
KrvskpEW	36	0.0170	22	0.0357	28
Lymphography	18	0.0050	9	0.1034	8
M-of-N	13	0.0046	6	0.0210	8
PenglungEW	325	0.0007	22	0.0671	36
Sonar	60	0.0058	35	0.0491	12
SpectEW	22	0.0242	13	0.1503	8
Tic-tac-toe	9	0.1492	9	0.1986	7
Vote	16	0.0013	2	0.0514	3
WaveformEW	40	0.1525	28	0.1783	28
Wine	13	0.0023	3	0.0321	6
Zoo	16	0.0025	4	0.0533	6

The fitness score is a measure of the quality of a particular solution. The lower the fitness score, the better the solution is. The best fitness score is the lowest score obtained among the 10 runs. Conversely, the worst fitness score is the highest score obtained among the 10 runs. Table 4 presents the results of feature selection performance using MMO-KNN over 10 runs for all 18 UCI benchmarking datasets. Each dataset is accompanied by its corresponding best and worst fitness scores, indicating the quality of the selected feature subsets. Additionally, Table 4 includes the counts of features selected for both the best and worst fitness scores.

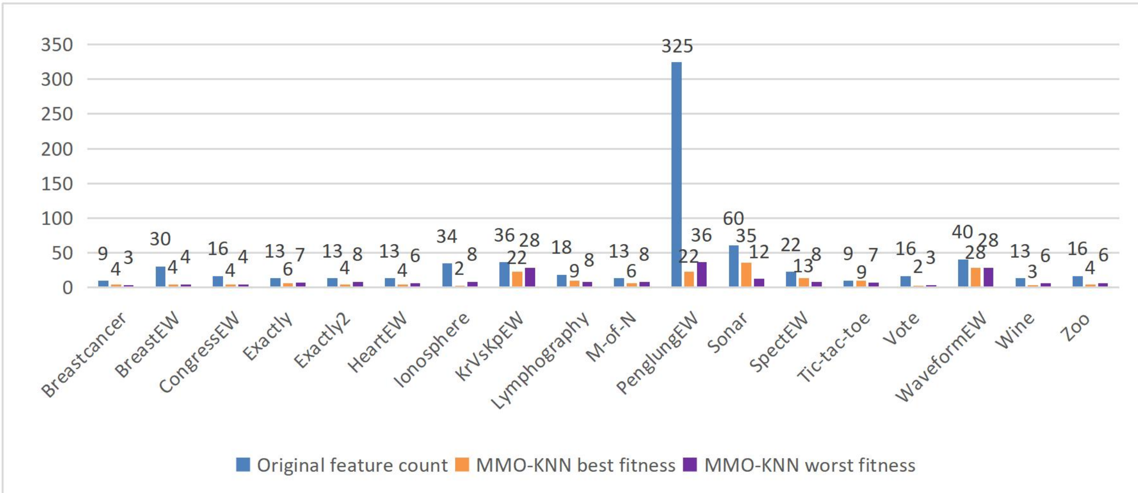

Figure 21. Comparison of the original number of features and MMO-KNN selected features count for the best and worst fitness across all 18 UCI benchmarking datasets

Moreover, the comparison of the original number of features and the selected features by MMO-KNN for best and worst fitness is visually illustrated in Figure 21.

Table 5. MMO-SVMs selected features Count based on best and worst fitness Scores over 10 runs

	MMO-SVM			
Dataset	Best fitness score	Best feature count	Worst fitness score	Worst feature count
Breastcancer	0.0101	3	0.0454	3
BreastEW	0.0267	2	0.0705	3
CongressEW	0.0019	3	0.0594	4
Exactly	0.0046	4	0.0796	7
Exactly2	0.2255	10	0.2323	8
HeartEW	0.0596	6	0.1681	4
Ionosphere	0.0018	6	0.0439	5
KrvskpEW	0.0119	26	0.0330	24
Lymphography	0.0347	3	0.1331	2
M-of-N	0.0046	6	0.0054	7
PenglungEW	0.0006	21	0.1325	15
Sonar	0.0033	20	0.0725	11
SpectEW	0.0211	6	0.1498	7
Tic-tac-toe	0.0873	9	0.1326	8
Vote	0.0019	3	0.0520	4
WaveformEW	0.1149	28	0.1434	27
Wine	0.0019	3	0.0588	5
Zoo	0.0025	4	0.0526	5

Also, Table 5 provides a comprehensive overview of the performance of feature selection of the MMO-SVM. This evaluation also spans 10 separate runs across all 18

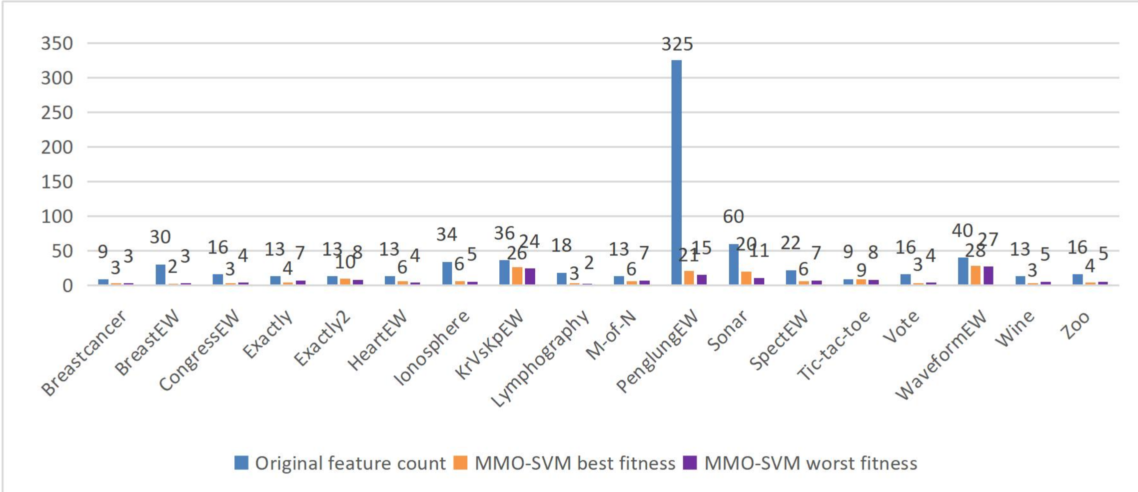

Figure 22. Comparison of the original number of features and MMO-SVM selected features count for the best and worst fitness across all 18 UCI benchmarking datasets

UCI benchmarking datasets. For each dataset, both the best and worst fitness scores achieved during these runs have been documented. Alongside the best and worst fitness scores documented in Table 5 for each dataset, the number of features corresponding to each fitness score has also been included. Moreover, the comparison of the original number of features and the selected features by MMO-SVM for best and worst fitness is visually illustrated in Figure 22.

In understanding the Monarch Butterfly Optimization (MBO) algorithm, it is essential to recognize the direct relationship between the fitness score and the quality of the selected feature subset. The optimization process of the FS algorithm revolves around enhancing this score, which entails selecting an optimal number of features that contribute most effectively to it.

Consider the Breastcancer dataset as an example. The Breastcancer dataset is a collection of instances representing cellular characteristics of breast tissue. Each instance in the dataset is categorized as either benign or malignant tumors (Wolberg, 1992).

Table 6. MMO-KNN's feature selection performance on Breastcancer dataset

Dataset	Original features	Best fitness selected features	Worst fitness selected features
Breastcancer	Clump thickness	Uniformity of cell size	Uniformity of cell size
	Uniformity of cell size	Marginal adhesion	Single Epithelial cell size
	Uniformity of cell shape	Bare nuclei	Mitoses
	Marginal adhesion	Normal nucleoli	
	Single epithelial cell size		
	Bare nuclei		
	Bland chromatin		
	Normal nucleoli		
	Mitoses		

The Breastcancer dataset serves as an example shown in Table 6, illustrating the original features, the selected features based on the best fitness, and the selected features based on the worst fitness for MMO-KNN results.

Table 7. MMO-SVM's feature selection performance on Breastcancer dataset

Dataset	Original features	Best fitness selected features	Worst fitness selected features
Breastcancer	Clump Thickness	Uniformity of cell size	Uniformity of Cell Size
	Uniformity of Cell Size	Uniformity of cell shape	Single Epithelial Cell Size
	Uniformity of Cell Shape	Bland chromatin	Mitoses
	Marginal Adhesion		
	Single Epithelial Cell Size		
	Bare Nuclei		
	Bland Chromatin		
	Normal Nucleoli		
	Mitoses		

Additionally, Table 7 shows the result of MMO-SVM FS for the *Breastcancer* dataset, which also include the selected features based on best and worst fitness.

Notice the variation in features selected by MMO-KNN and MMO-SVM for the best fitness, while they remain consistent for the worst fitness. These differences arise from the distinct methodologies and underlying principles employed by each classification algorithm. Although both MMO-KNN and MMO-SVM share the goal of identifying the most informative subset of features for classification tasks, they employ different strategies to achieve this objective. Consequently, variations occur in the specific features chosen by each algorithm.

Table 8. Comparison of the standard KNN and the proposed MMO-KNN concerning classification accuracy based on best and worst fitness

Dataset	KNN's averaged classification accuracy	Best fitness MMO-KNN's classification accuracy	Worst fitness MMO-KNN's classification accuracy
Breastcancer	0.6086	1.0	0.9643
BreastEW	0.9070	0.9912	0.9386
CongressEW	0.9253	1.0	0.9425
Exactly	0.7265	1.0	0.98
Exactly2	0.7295	0.81	0.785
HeartEW	0.7019	0.9259	0.8333
Ionosphere	0.8414	1.0	0.9429
KrvskpEW	0.9613	0.989	0.9718
Lymphography	0.7667	1.0	0.9
M-of-N	0.8895	1.0	0.985
PenglungEW	0.7993	1.0	0.9333
Sonar	0.8119	1.0	0.9524
SpectEW	0.7981	0.9815	0.8519
Tic-tac-toe	0.8354	0.8594	0.8073
Vote	0.9167	1.0	0.95
WaveformEW	0.8063	0.853	0.827
Wine	0.7139	1.0	0.9722
Zoo	0.87	1.0	0.95

The classification accuracy between MMO-KNN and the standard KNN classifier, as well as between MMO-SVM and the standard SVM classifier, is now compared. Consistent improvements in classification accuracy are evident for MMO-KNN across various datasets, as detailed in Table 8.

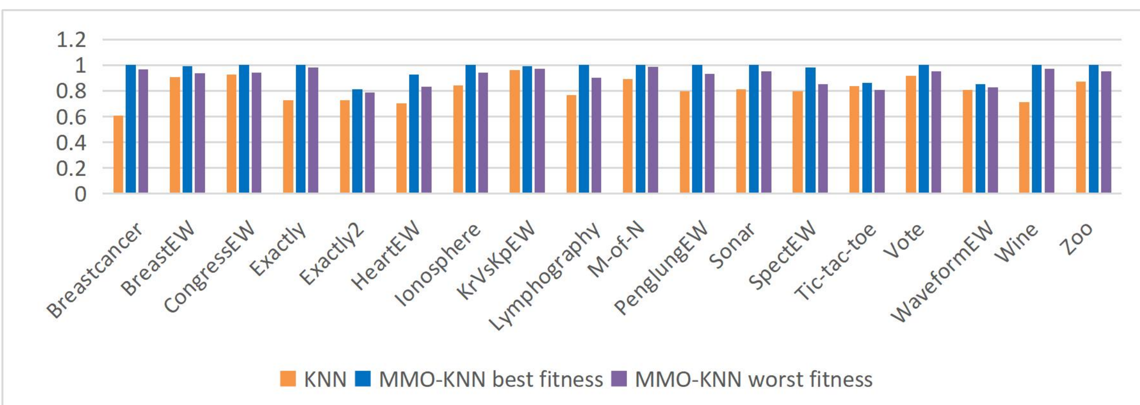

Figure 23. Comparison of standard KNN and proposed MMO-KNN in terms of classification accuracy based on best and worst fitness

These findings are further supported by visual representations in Figure 23, offering additional insight into the performance disparities between MMO-KNN and its standard counterpart. The best fitness scenarios often yield near-perfect or perfect classification accuracy, indicating the ability of MMO-KNN to identify and utilize the most relevant features for the classification task. Even in less favorable scenarios represented by the worst fitness cases, MMO-KNN maintains higher accuracy than its standard counterpart, highlighting the robustness of MMO as a feature selection method for KNN classifiers.

Table 9. Comparison of the standard SVM and the proposed MMO-SVM concerning classification accuracy based on best and worst fitness

Dataset	SVM's averaged classification accuracy	Best fitness MMO-SVM's classification accuracy	Worst fitness MMO-SVM's classification accuracy
Breastcancer	0.6571	0.9928	0.9571
BreastEW	0.9105	0.9737	0.9298
CongressEW	0.9563	1.0	0.9425
Exactly	0.712	1.0	0.925
Exactly2	0.761	0.78	0.77
HeartEW	0.6574	0.9444	0.8333
Ionosphere	0.9443	1.0	0.9571
KrvskpEW	0.9757	0.9953	0.9734
Lymphography	0.7833	0.9667	0.8667
M-of-N	1.0	1.0	1.0
PenglungEW	0.7533	1.0	0.8667
Sonar	0.8286	1.0	0.9286
SpectEW	0.8296	0.9815	0.8519
Tic-tac-toe	0.8927	0.9218	0.875
Vote	0.9533	1.0	0.95
WaveformEW	0.8663	0.891	0.862
Wine	0.7133	1.0	0.9444
Zoo	0.94	1.0	0.95

Similarly, Table 9 demonstrates significant improvements in classification accuracy with MMO-SVM than the standard SVM classifier. In scenarios where the best fitness is achieved, the accuracy levels are consistently higher, highlighting the effectiveness of MMO-SVM in utilizing relevant features to enhance classification performance. Additionally, in situations where the feature selection is suboptimal, MMO-SVM exhibits greater robustness compared to the standard SVM, ensuring more consistent and reliable classification results across various datasets. These findings are further corroborated by the visual representation, which provides a clearer depiction of the performance differences between MMO-SVM and the standard SVM. This offer

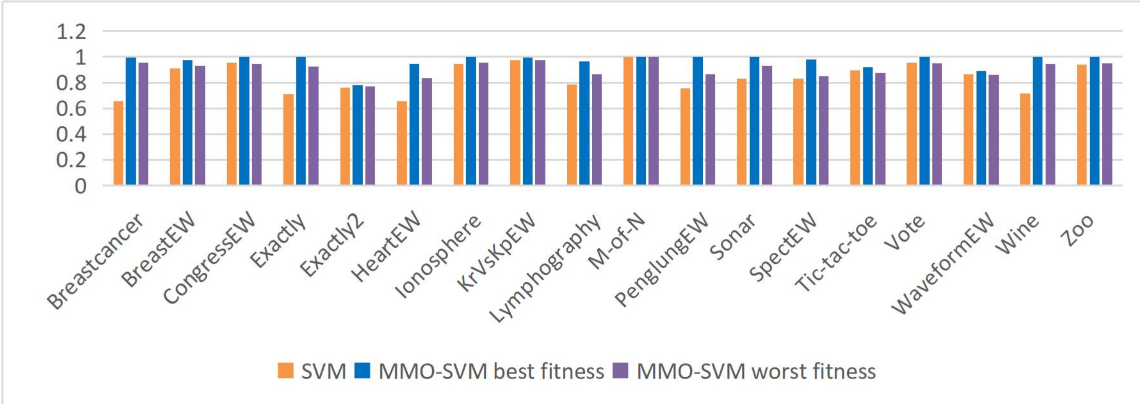

Figure 24. Comparison of standard SVM and proposed MMO-SVM in terms of classification accuracy based on best and worst fitness

deeper insights into the comparative advantages of MMO-SVM, as shown in Figure 24.

Overall, the comparison underscores the significant performance improvements facilitated by MMO as a feature selection method for both KNN and SVM classifiers. By leveraging MMO, classifiers achieved more robust and consistent enhanced performance, thereby enhancing their utility in various classification tasks.

Comparison of MMO and MA-HS

The proposed MMO method is set to undergo comparison with MA-HS, which stands for Mayfly Algorithm-Harmony Search, introduced by Bhattacharyya et al. (2020). While MMO is a fusion of Monarch Butterfly Optimization (MBO) and Mayfly Algorithm (MA), MA-HS integrates Mayfly Algorithm with Harmony Search. Bhattacharyya et al. (2020) conducted a comprehensive study comparing MA-HS with 12 other state-of-the-art metaheuristic feature selection (FS) methods. Remarkably, MA-HS outperformed all others, emerging as the top performer across 18 benchmark UCI datasets, which confirms its effectiveness. Given this notable outcome, the proposed MMO method will undergo rigorous experimentation using the same 18 benchmark UCI datasets, aiming to discern its comparative performance against MA-HS. This evaluation will shed light on the potential strengths and areas for improvement of MMO in tackling FS challenges across various datasets.

Table 10. Comparison of MMO-KNN and MA-HS-KNN concerning selected features count across best fitness, worst fitness, and averages for all datasets

Dataset	Original no. of features	MMO-KNN			MA-HS-KNN		
		Best fitness selected features count (BFSFC)	Worst fitness selected features count (WFSFC)	Average selected features count (ASFC)	Best fitness selected features count (BFSFC)	Worst fitness selected features count (WFSFC)	Average selected features count (ASFC)
Breastcancer	9	4	**3**	4.2	**2**	**3**	**2.4**
BreastEW	30	**4**	4	**3.5**	6	**3**	5.7
CongressEW	16	**4**	**4**	4.9	**4**	**4**	**2.9**
Exactly	13	**6**	**7**	6.7	**6**	**7**	**6.2**
Exactly2	13	**4**	8	7	**4**	**1**	**1.7**
HeartEW	13	**4**	6	4.7	**4**	**4**	**4.2**
Ionosphere	34	**2**	8	**4.1**	**2**	**5**	4.4
KrVsKpEW	36	22	28	23.1	**17**	**24**	**17.3**
Lymphography	18	9	8	7	**4**	**4**	**5.9**
M-of-N	13	**6**	8	6.9	**6**	**7**	**6.4**
PenglungEW	325	**22**	**36**	**32**	73	79	77.6
Sonar	60	35	**12**	**13.1**	**19**	15	16.5
SpectEW	22	13	8	8.1	**6**	**7**	**7.2**
Tic-tac-toe	9	**9**	7	7.6	**9**	**5**	**5.3**
Vote	16	**2**	**3**	3.6	**2**	**3**	**2.8**
WaveformEW	40	28	28	27.3	**22**	**22**	**16.9**
Wine	13	**3**	6	**3.7**	4	**2**	4.1
Zoo	16	**4**	**6**	**4.4**	**4**	9	5.2
Average BFSFC rank		1.7			**1.2**		
Assigned BFSFC rank		2			1		
Average WFSFC rank			1.6			**1.2**	
Assigned WFSFC rank			2			1	
Average ASFC rank				1.7			**1.3**
Assigned ASFC rank				2			1

The MA-HS-KNN method consistently selects fewer features on average compared to the MMO-KNN method, suggesting that it is more efficient in reducing features to achieve the best fitness, as illustrated in Table 10. Similarly, for the worst fitness, the MA-HS-KNN method selects fewer features on average, indicating better consistency and potentially more robustness in feature selection. Across all selected feature counts, the MA-HS-KNN method shows a more efficient feature selection process with fewer features needed on average. However, it has already been established that the objective of FS is to select the optimal number of features needed where the learning model's performance is not compromised and maintained as highly as possible.

Table 11. Comparison of MMO-KNN and MA-HS-KNN concerning classification accuracy across best fitness, worst fitness, and averages for all datasets

Dataset	MMO-KNN			MA-HS-KNN		
	Best fitness classification accuracy (BFCA)	Worst fitness classification accuracy (WFCA)	Average classification accuracy (ACA)	Best fitness classification accuracy (BFCA)	Worst fitness classification accuracy (WFCA)	Average classification accuracy (ACA)
Breastcancer	1.0	**0.9643**	0.9857	0.9929	**0.9643**	0.9757
BreastEW	**0.9912**	**0.9386**	**0.9649**	0.9825	0.9211	0.9596
CongressEW	**1.0**	**0.9425**	**0.9782**	**1.0**	**0.9425**	0.9701
Exactly	**1.0**	0.98	0.9935	**1.0**	**0.985**	**0.9985**
Exactly2	**0.81**	**0.785**	**0.7965**	**0.81**	0.76	0.769
HeartEW	**0.9259**	**0.8333**	**0.8833**	**0.9259**	0.8148	0.8778
Ionosphere	**1.0**	**0.9429**	**0.9729**	0.9714	0.9143	0.9457
KrVsKpEW	**0.989**	0.9718	**0.9817**	0.9859	**0.9828**	0.9764
Lymphography	**1.0**	**0.9**	**0.95**	0.9667	0.8667	0.9333
M-of-N	**1.0**	0.985	0.9975	**1.0**	**1.0**	**1.0**
PenglungEW	**1.0**	**93.33**	**0.9733**	**1.0**	0.8	0.9333
Sonar	**1.0**	**0.9524**	**0.9762**	0.9762	0.9048	0.9405
SpectEW	**0.9815**	**0.8519**	**0.9037**	0.963	0.8333	0.8907
Tic-tac-toe	**0.8594**	**0.8073**	**0.8380**	0.8542	0.7813	0.8229
Vote	**1.0**	**0.95**	**0.9883**	**1.0**	**0.95**	0.9817
WaveformEW	**0.853**	**0.827**	**0.8363**	0.838	**0.827**	0.8246
Wine	**1.0**	**0.9722**	**0.9861**	**1.0**	0.9444	0.9833
Zoo	**1.0**	**0.95**	**0.9778**	**1.0**	**0.95**	0.975
Average BFCA rank	**1**			1.5		
Assigned BFCA rank	**1**			2		
Average WFCA rank		**1.2**			1.6	
Assigned WFCA rank		**1**			2	
Average ACA rank			**1.1**			1.9
Assigned ACA rank			**1**			2

Even though MA-HS-KNN had selected fewer features compared to MMO-KNN, MMO-KNN had achieved higher classification accuracy, as shown in Table 11. MMO-KNN achieves higher classification accuracy for the best fitness scenarios, indicating it is more effective in finding the optimal feature subset for maximum accuracy. In addition, MMO-KNN demonstrates better performance for worst-case scenarios, indicating more robust and consistent performance. Moreover, MMO-KNN also shows superior performance in terms of average classification accuracy across all datasets.

Despite the consistent efficiency of MA-HS-KNN in the selection of features, the overall results suggest that MMO-KNN offers more consistent and reliable improvements in classification accuracy across diverse datasets. These findings are reflected in its higher averaged classification accuracy than MA-HS-KNN. Next, the performance of MMO-SVM and MA-HS-SVM is compared regarding feature selection and classification accuracy.

Table 12. Comparison of MMO-SVM and MA-HS-SVM concerning selected features count across best fitness, worst fitness, and averages for all datasets

Dataset	Original no. of features	MMO-SVM			MA-HS-SVM		
		Best fitness selected features count (BFSFC)	Worst fitness selected features count (WFSFC)	Average selected features count (ASFC)	Best fitness selected features count (BFSFC)	Worst fitness selected features count (WFSFC)	Average selected features count (ASFC)
Breastcancer	9	**3**	3	3.7	**3**	**2**	**2.4**
BreastEW	30	**2**	**3**	**3.8**	7	5	4.7
CongressEW	16	**3**	**4**	5.1	**3**	**4**	**2.2**
Exactly	13	6	**7**	**6.2**	6	8	7.2
Exactly2	13	10	8	7.2	**1**	**1**	**1**
HeartEW	13	**6**	**4**	**4.8**	**6**	7	4.9
Ionosphere	34	**6**	**5**	**6.6**	8	10	8.9
KrVsKpEW	36	26	24	25	**18**	**16**	**14.8**
Lymphography	18	**3**	**2**	**4.2**	6	3	5.8
M-of-N	13	**6**	**7**	**6.4**	**6**	**7**	**6.4**
PenglungEW	325	**21**	**15**	**25.7**	103	69	75.9
Sonar	60	**20**	**11**	**12.1**	23	25	21.6
SpectEW	22	**6**	7	**5.6**	8	**5**	6.5
Tic-tac-toe	9	**9**	8	8.3	**9**	**7**	**7.7**
Vote	16	**3**	4	4.9	**3**	**3**	**2.8**
WaveformEW	40	28	27	28.6	**25**	**18**	**19.4**
Wine	13	**3**	5	3.9	5	**2**	**3.3**
Zoo	16	**4**	**5**	**5.1**	**4**	7	5.8
	Average BFSFC rank	**1.2**			1.4		
	Assigned BFSFC rank	**1**			2		
	Average WFSFC rank		**1.4**			**1.4**	
	Assigned WFSFC rank		**1**			**1**	
	Average ASFC rank			**1.4**			1.5
	Assigned ASFC rank			**1**			2

The MMO-SVM method slightly outperforms MA-HS-SVM in selecting fewer features on average for achieving the best fitness, as shown in Table 12. Also, both methods perform equally, indicating similar consistency in feature selection for the worst fitness scenarios. The MMO-SVM method also slightly outperforms MA-HS-SVM in terms of efficiency in feature selection on average, suggesting it is marginally more efficient.

This time, MMO-SVM demonstrates a slight edge over MA-HS-SVM in terms of selecting fewer features for best and average scenarios. Additionally, both methods are equally consistent in selecting features for the worst fitness scenarios.

Table 13. Comparison of MMO-SVM and MA-HS-SVM concerning selected features count across best fitness, worst fitness, and averages for all datasets

Dataset	MMO-SVM			MA-HS-SVM		
	Best fitness classification accuracy (BFCA)	Worst fitness classification accuracy (WFCA)	Average classification accuracy (ACA)	Best fitness classification accuracy (BFCA)	Worst fitness classification accuracy (WFCA)	Average classification accuracy (ACA)
Breastcancer	**0.9929**	**0.9571**	**0.9807**	**0.9929**	0.95	0.9757
BreastEW	**0.9737**	**0.9298**	**0.9605**	**0.9737**	0.9211	0.9509
CongressEW	**1.0**	**0.9425**	**0.9759**	**1.0**	**0.9425**	0.9644
Exactly	**1.0**	**0.925**	**0.9725**	0.98	0.825	0.8985
Exactly2	**0.78**	**0.77**	**0.773**	0.76	0.76	0.76
HeartEW	**0.9444**	0.8333	**0.8926**	**0.9444**	**0.8519**	0.8889
Ionosphere	**1.0**	**0.9571**	**0.9886**	**1.0**	**0.9571**	0.9814
KrVsKpEW	**0.9953**	**0.9734**	**0.9881**	0.9937	0.9671	0.9778
Lymphography	**0.9667**	**0.8667**	**0.9233**	**0.9667**	**0.8667**	0.92
M-of-N	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
PenglungEW	**1.0**	**0.8667**	**0.96**	**1.0**	0.7333	0.88
Sonar	**1.0**	**0.9286**	**0.9571**	**1.0**	0.881	0.9333
SpectEW	**0.9815**	**0.8519**	**0.8944**	0.963	0.8333	0.8852
Tic-tac-toe	**0.9219**	**0.875**	**0.8964**	**0.9219**	0.8646	0.8922
Vote	**1.0**	**0.95**	**0.985**	**1.0**	0.9333	0.9717
WaveformEW	0.891	**0.862**	**0.8756**	**0.894**	0.848	0.8633
Wine	**1.0**	**0.9444**	**0.9694**	**1.0**	0.9167	0.9611
Zoo	**1.0**	**0.95**	**0.98**	**1.0**	**0.95**	**0.98**
Average BFCA rank	**1.06**			1.22		
Assigned BFCA rank	**1**			2		
Average WFCA rank		**1.06**			1.67	
Assigned WFCA rank		**1**			2	
Average ACA rank			**1**			1.9
Assigned ACA rank			**1**			2

For the second time, the MMO-SVM method generally offers better classification accuracy than MA-HS-SVM. MMO-SVM achieves higher accuracy in best fitness, worst fitness, and average scenarios, providing more reliable and consistent performance. These results are shown in Table 13.

The discussion underscores the nuanced yet critical differences between MMO and MA-HS in feature selection and classification accuracy. MA-HS excels in feature selection efficiency by selecting fewer features on average, which can be beneficial for reducing model complexity and computational costs. While this observation does not always lead to higher classification accuracy, it is worth noting that MMO exhibits better overall classification accuracy, robustness, and reliability. This suggests their effectiveness in achieving high performance across diverse datasets. The findings underscore that the quality and relevance of selected features play a more critical role in classification accuracy than sheer quantity alone.

Time Complexity

To analyze the Monarch Mayfly Optimization (MMO), the core components and their time complexities will be broken down. The critical operations within the main MMO loop and their contributions to the overall complexity will be considered. The following assumptions simplify the analysis: the maximum number of iterations the MMO performs is $MaxIter$, represented as T, the total population size is N, and the number of features or dimensions each mayfly consists of is D. The subpopulation $NP1$ is $N1$, and the subpopulation $NP2$ is $(N - N1)$.

The MMO begins with updating the velocity limit to ensure that the velocities of mayflies are constrained within reasonable bounds. This operation has a time complexity of $O(N)$ since it considers the total population of mayflies. The MMO then proceeds to the $NP1$ subpopulation by updating the global best ($gbest$) and personal best ($pbest$) positions, each with a time complexity of $O(1)$. Next, the velocities of $NP1$ male and female mayflies are updated, with a time complexity of $O(N1)$. A $Lévy$ flight is performed for female mayflies in $NP1$, with a time complexity of $O(N1)$. Following this, the Migration Operator is applied to $NP1$ male mayflies, considering each mayfly and its dimensions, resulting in a time complexity of $O(N1 \times D)$. Similarly, the Adjusting

Operator is applied to $NP1$ female mayflies, with the same time complexity of $O(N1 \times D)$.

The fitness scores of $NP1$ male and female mayflies are then calculated using a classifier, shown in line 13 of Figure 17. The time complexity of this operation depends on the classifier used, such as KNN or SVM, each having its time complexities. For this analysis, the classifier's time complexity will be denoted as the $O(Classifier_{timeComplexity})$. The $NP1$ optimization continues sorting the $NP1$ male and female mayflies' positions, velocities, and fitness using the Quicksort algorithm, shown in line 14 of Figure 17, with a time complexity of $O(N \log N)$. Crossover and Mutation operations are then performed to generate and mutate two offspring, with each operation considering each dimension of each mayfly in $NP1$, resulting in a combined time complexity of $O(N1 \times D)$.

Next, the worst-performing mayflies in $NP1$ are replaced with the best new offspring generated, an operation with a time complexity of $O(1)$. The positions of $NP1$ male and female mayflies are then updated, which has a time complexity of $O(N1)$. Elitism is applied to preserve elite individuals and replace the worst individuals with elite ones, with a time complexity of $O(1)$. The gravity and nuptial dance parameters are then updated for $NP1$, with a time complexity of $O(1)$. The optimization of $NP1$ ends with an update to the $gbest$, which has a time complexity of $O(1)$. Hence, the time complexity for the $NP1$ optimization process is $O(N \log N + Classifier_{timeComplexity})$. Since $N \log N$ is the dominant term in the time complexity expression, and also considering the complexity of the classifier.

The $NP2$ subpopulation undergoes the same processes as $NP1$. However, $NP2$ iterates for a maximum number of iterations T, with a counter reset when a new $gbest$ is better than the current $gbest$, or incremented otherwise. The total number of iterations can be represented by cT, where $c \geq 1$. The counter is reset based on the performance of the $gbest$, and the factor c reflects how many times the counter may be reset during the iterative process. The MMO proceeds to the $NP2$ subpopulation by updating the $gbest$ and $pbest$, both with time complexities of $O(1)$. The velocities of $NP2$ male and female mayflies are updated, with a time complexity of $O(N - N1)$. A $Lévy$ flight is performed for female mayflies in $NP2$, with a time complexity of $O(N - N1)$. The Migration Operator is applied to $NP2$ male mayflies, with a time complexity of $O((N - N1) \times D)$, followed by the Adjusting Operator for $NP2$ female mayflies, also with a time complexity of $O((N - N1) \times D)$. The fitness scores of $NP2$ mayflies are then calculated using a classifier, shown in line 30 of Figure 17, which has a time complexity of $O(Classifier_{timeComplexity})$.

The $NP2$ optimization continues sorting the $NP2$ mayflies using the Quicksort algorithm, which has a time complexity of $O(N\ log\ N)$, as shown in line 31 of Figure 17. The new $gbest$, is then compared to the current $gbest$, with the counter being reset or incremented accordingly. The Crossover and Mutation operations are performed for $NP2$, with a time complexity of $O((N - N1) \times D)$. The worst-performing mayflies in $NP2$ are replaced with the best new offspring, with a time complexity of $O(1)$. The positions of $NP2$ male and female mayflies are updated, which has a time complexity of $O(N - N1)$. Elitism is applied, with a time complexity of $O(1)$, followed by updates to the gravity and nuptial dance parameters, each with a time complexity of $O(1)$. The optimization of

$NP2$ ends with an update to the $gbest$, with a time complexity of $O(1)$. Thus, the time complexity for the $NP2$ optimization process is $O(cT \times (N \log N + Classifier_{timeComplexity}))$.

The $NP1$ and $NP2$ are performed T times. Therefore, the time complexity of MMO is $O(T \times [(N \log N + Classifier_{timeComplexity}) + cT \times (N \log N + Classifier_{timeComplexity})]) = O(T \times (cT \times (N \log N + Classifier_{timeComplexity}))) = O(cT^2 \times N \log N + cT^2 \times Classifier_{timeComplexity})) \approx O(N \log N + Classifier_{timeComplexity})$. The time complexity implies that the classifier affects the overall time complexity of the MMO. For instance, when the classifier's time complexity is worse than that of the MMO optimization process alone, the MMO process becomes slower.

CHAPTER V

SUMMARY, CONCLUSION, AND RECOMMENDATION

This final chapter provides a comprehensive summary of the research findings, draws conclusions based on these findings, and offers recommendations for future studies.

Summary of Findings

This section summarizes the main findings from the assessment of the proposed feature selection method, Monarch Mayfly Optimization (MMO). MMO's performance as a wrapper was evaluated with KNN and SVM classifiers across 18 UCI benchmarking datasets, examining how the number of selected features varies across fitness scores and its impact on classification accuracy. The experimental results showed that the selected optimal feature subset is different for MMO-KNN and MMO-SVM. Additionally, The classification accuracy of standard classifiers such as KNN and SVM is greatly improved when MMO is used for feature selection. Moreover, while MA-HS selected fewer features, MMO achieved higher classification accuracy using KNN and SVM classifiers despite selecting more features compared to MA-HS. Furthermore, the time complexity of MMO is approximately $O(N \, log \, N \, + Classifier_{timeComplexity})$.

Conclusion

The study shows that the hybridization of metaheuristic algorithms can be effective in solving the feature selection problem. Particularly, the variation in the selection of optimal feature subsets underscores the unique selection criteria and decision-making process inherent to each classifier. In addition, MMO is effective in identifying relevant features, thereby optimizing the performance of KNN and SVM classifiers. Moreover, prioritizing the quality and relevance of selected features is more significant than sheer quantity when aiming to improve classification accuracy. Furthermore, wrapper method like MMO is affected by the classifier's time complexity.

Recommendations

Based on the study's analysis and findings, key recommendations are proposed for advancing the Monarch Mayfly Optimization (MMO) algorithm. These suggestions aim to boost its effectiveness, robustness, and computational efficiency in feature selection and classification tasks. The insights highlight MMO's potential to enhance classification accuracy and streamline feature sets across various datasets. To tackle identified challenges and capitalize on these strengths, the following directions for future research and practical use are suggested.

First, experiment on the effect of the ratio of males and females in each subpopulation. For example, there might be more males than females in a given subpopulation, while in other cases, the opposite might be true. By varying the gender ratio across different subpopulations, we can observe and analyze how these differences impact the overall dynamics and outcomes of the MMO process. This approach allows us to explore a range of scenarios and better understand the influence of gender composition on the behaviors and characteristics of the subpopulations.

Second, conduct feature selection analysis across datasets. A detailed analysis of how the number of features selected by MMO varies across best and worst fitness scenarios for each specific dataset. This will help in understanding the adaptability and performance of MMO in various contexts. Understanding the variability in feature selection can guide the customization of MMO for different datasets, improving its generalizability and effectiveness in practical applications.

Third, extend the comparative analysis by including more classifiers and datasets. Researchers can better understand how MMO performs across different contexts and uncover why it outperforms or underperforms compared to standard methods. Demonstrating consistent improvements in classification accuracy with MMO can promote its adoption in fields requiring high precision and reliability. For example, fields such as medical diagnostics and financial forecasting.

Lastly, a comprehensive comparison of MMO with other hybrid methods can highlight its strengths and potential areas for improvement, fostering innovation in feature selection techniques.

REFERENCES

Agrawal, P., Abutarboush, H. F., Ganesh, T., & Mohamed, A. W. (2021). Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019). Ieee Access, 9, 26766-26791.

Al-Wajih, R., Abdulkadir, S. J., Aziz, N., Al-Tashi, Q., & Talpur, N. (2021). Hybrid binary grey wolf with Harris hawks optimizer for feature selection. IEEE Access, 9, 31662-31677.

Arora, S., Singh, S., Singh, S., & Sharma, B. (2014, December). Mutated firefly algorithm. In 2014 International Conference on Parallel, Distributed and Grid Computing (pp. 33-38). IEEE.

Bhattacharyya, T., Chatterjee, B., Singh, P. K., Yoon, J. H., Geem, Z. W., & Sarkar, R. (2020). Mayfly in harmony: A new hybrid meta-heuristic feature selection algorithm. IEEE Access, 8, 195929-195945.

Bimurat, Z., Kim, Y., Ismailova, R., & Sagindykov, B. (2023). METHODS OF NAVIGATING ALGORITHMIC COMPLEXITY: BIG-OH AND SMALL-OH NOTATIONS. Scientific Journal of Astana IT University, 160-181.

Eid, M. M., El-kenawy, E. S. M., & Ibrahim, A. (2021, March). A binary sine cosine-modified whale optimization algorithm for feature selection. In 2021 National Computing Colleges Conference (NCCC) (pp. 1-6). IEEE.

Geem, Z. W., Kim, J. H., & Loganathan, G. V. (2001). A new heuristic optimization algorithm: harmony search. simulation, 76(2), 60-68.

Ghosh, K. K., Singh, P. K., Hong, J., Geem, Z. W., & Sarkar, R. (2020). Binary social mimic optimization algorithm with x-shaped transfer function for feature selection. IEEE Access, 8, 97890-97906.

Guo, S. S., Wang, J. S., & Guo, M. W. (2020). Z-shaped transfer functions for binary particle swarm optimization algorithm. Computational Intelligence and Neuroscience, 2020.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. Journal of machine learning research, 3(Mar), 1157-1182.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362.

Islam, M. J., Li, X., & Mei, Y. (2017). A time-varying transfer function for balancing the exploration and exploitation ability of a binary PSO. Applied Soft Computing, 59, 182-196.

Jović, A., Brkić, K., & Bogunović, N. (2015, May). A review of feature selection methods with applications. In 2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO) (pp. 1200-1205). Ieee.

Kareem, S. S., Mostafa, R. R., Hashim, F. A., & El-Bakry, H. M. (2022). An effective feature selection model using hybrid metaheuristic algorithms for iot intrusion detection. Sensors, 22(4), 1396.

Kennedy, J., & Eberhart, R. (1995, November). Particle swarm optimization. In Proceedings of ICNN'95-international conference on neural networks (Vol. 4, pp. 1942-1948). IEEE.

Liu, H., & Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. IEEE Transactions on knowledge and data engineering, 17(4), 491-502.

Mirjalili, S. M. S. M., Mirjalili, S. M., & Lewis, A. (2014). Grey Wolf Optimizer Adv Eng Softw 69: 46–61.

Mirjalili, S., Zhang, H., Mirjalili, S., Chalup, S., & Noman, N. (2020). A novel U-shaped transfer function for binary particle swarm optimisation. In Soft Computing for Problem Solving 2019: Proceedings of SocProS 2019, Volume 1 (pp. 241-259). Springer Singapore.

Mohamed, A. W., Hadi, A. A., & Mohamed, A. K. (2020). Gaining-sharing knowledge based algorithm for solving optimization problems: a novel nature-inspired algorithm. International Journal of Machine Learning and Cybernetics, 11(7), 1501-1529.

Nezamabadi-pour, H., Rostami-Shahrbabaki, M., & Maghfoori-Farsangi, M. (2008). Binary particle swarm optimization: challenges and new solutions. CSI J Comput Sci Eng, 6(1), 21-32.

Olorunda, O., & Engelbrecht, A. P. (2008, June). Measuring exploration/exploitation in particle swarms using swarm diversity. In 2008 IEEE congress on evolutionary computation (IEEE world congress on computational intelligence) (pp. 1128-1134). IEEE.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.

Reback, J., McKinney, W., Van Den Bossche, J., Augspurger, T., Cloud, P., Klein, A., ... & Seabold, S. (2020). pandas-dev/pandas: Pandas 1.0. 5. Zenodo.

Sharma, M., & Kaur, P. (2021). A comprehensive analysis of nature-inspired meta-heuristic techniques for feature selection problem. Archives of Computational Methods in Engineering, 28, 1103-1127.

Sheikh, K. H., Ahmed, S., Mukhopadhyay, K., Singh, P. K., Yoon, J. H., Geem, Z. W., & Sarkar, R. (2020). EHHM: Electrical harmony based hybrid meta-heuristic for feature selection. IEEE Access, 8, 158125-158141.

Ting, T. O., Yang, X. S., Cheng, S., & Huang, K. (2015). Hybrid metaheuristic algorithms: past, present, and future. Recent advances in swarm intelligence and evolutionary computation, 71-83

UC Irvine Machine Learning Repository. (n.d.). UC Irvine Machine Learning Repository. Retrieved from https://archive.ics.uci.edu/ml/index.php

Van Rossum, G. (2020). The python library reference, release 3.8. 2. Python Software Foundation, 36.

Wang, G. G., Deb, S., & Cui, Z. (2019). Monarch butterfly optimization. Neural computing and applications, 31, 1995-2014.

Wolberg,WIlliam. (1992). Breast Cancer Wisconsin (Original). UCI Machine Learning Repository. https://doi.org/10.24432/C5HP4Z.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. IEEE transactions on evolutionary computation, 1(1), 67-82.

Zervoudakis, K., & Tsafarakis, S. (2020). A mayfly optimization algorithm. Computers & Industrial Engineering, 145, 106559.

APPENDICES

APPENDIX A

MMO-KNN SOURCE CODE

```python
"""
Monarch Mayfly Optimization (MMO) Algorithm for Feature Selection
=================================================================
This implementation of the Monarch Mayfly Optimization (MMO) algorithm
is a hybrid of the Monarch Butterfly Optimization (MBO) and the Mayfly Algorithm (MA).

Sources and Acknowledgements:
-----------------------------
1. Monarch Butterfly Optimization (MBO):
   - Original implementation in Python by Justin van Zyl.
   - Based on the study:
     Wang G., Deb S., Cui Z., "Monarch Butterfly Optimization," Neural Comput & Applic 31:1995-2014.
     doi: 10.1007/s00521-015-1923-y.
   - Key operations utilized: Migration Operator, Adjusting Operator, and Elitism.

2. Mayfly Algorithm (MA):
   - Extracted from the hybrid feature selection study:
     Bhattacharyya, T., Chatterjee, B., Singh, P. K., Yoon, J. H., Geem, Z. W., & Sarkar, R. (2020).
     "Mayfly in harmony: A new hybrid meta-heuristic feature selection algorithm," IEEE Access, 8, 195929-195945.
   - The study hybridized MA with Harmony Search (HS). The MA part is used in this MMO hybrid.

Disclaimer:
-----------
This code is a hybrid implementation of the aforementioned algorithms and combines elements from both
to create the MMO algorithm for the purpose of feature selection. Full credit goes to the original authors
for their contributions.
"""
import numpy as np
import pandas as pd
import math
import random
from time import process_time
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

start_time = process_time()

# Load dataset
df = pd.read_csv('Breastcancer.csv')
tot_features = len(df.columns) - 1
x = df[df.columns[:tot_features]]
y = df[df.columns[-1]]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stratify=y)

# Train classifier using original dataset
_classifier = KNeighborsClassifier(n_neighbors=5)
_classifier.fit(x_train, y_train)
predictions = _classifier.predict(x_test)
total_acc = accuracy_score(y_true=y_test, y_pred=predictions)
total_error = 1 - total_acc
total_features = tot_features
total_acc

# Controlling parameters
swarm_size = 20
max_iterations = 20
alpha = 0.01
a1 = 3
a2 = 3.5
beta = 0.1
d = 3
fl = 3
l = 0.95
g = 1
delta = 0.9
lf_size = 1
adjusting_rate = 0.1
p = float(6/12)
```

```python
s_max = 0.02
gmax=9.8
gmin=6
max_neighbors = 20

# Population structure and Initialization
subpop_size = swarm_size // 2
NP1_male_swarm_vel = np.zeros((subpop_size, tot_features))
NP1_female_swarm_vel = np.zeros((subpop_size, tot_features))
NP2_male_swarm_vel = np.zeros((subpop_size, tot_features))
NP2_female_swarm_vel = np.zeros((subpop_size, tot_features))

NP1_male_swarm_pos = np.random.uniform(low=-1, high=1, size=(subpop_size, tot_features))
NP1_female_swarm_pos = np.random.uniform(low=-1, high=1, size=(subpop_size, tot_features))
NP2_male_swarm_pos = np.random.uniform(low=-1, high=1, size=(subpop_size, tot_features))
NP2_female_swarm_pos = np.random.uniform(low=-1, high=1, size=(subpop_size, tot_features))

gbest_fitness = 1000000
pbest_fitness = np.empty(swarm_size)
pbest_fitness.fill(np.inf)
pbest = np.zeros((swarm_size, tot_features))
gbest = np.zeros(tot_features)
NP1_male_fitness = np.empty(subpop_size)
NP1_female_fitness = np.empty(subpop_size)
NP2_male_fitness = np.empty(subpop_size)
NP2_female_fitness = np.empty(subpop_size)
NP1_vmax_male = np.empty(tot_features)
NP1_vmax_female = np.empty(tot_features)
NP2_vmax_male = np.empty(tot_features)
NP2_vmax_female = np.empty(tot_features)

# S-shaped transfer function
def transfer_func(velocity):
    s1 = np.abs(velocity) * 0.5 + 1
    s1 = (-velocity) / s1 + 0.5
    return s1

# Fitness function
def find_fitness(particle):
    features = [df.columns[i] for i, v in enumerate(transfer_func(particle)) if v >= 0.25]
    if not features:
        return 10000
    new_x_train = x_train[features].copy()
    new_x_test = x_test[features].copy()
    _classifier = KNeighborsClassifier(n_neighbors=5)
    _classifier.fit(new_x_train, y_train)
    predictions = _classifier.predict(new_x_test)
    acc = accuracy_score(y_true=y_test, y_pred=predictions)
    err = 1 - acc
    num_features = len(features)
    fitness = alpha * (num_features / total_features) + (1 - alpha) * err
    return fitness

# Levy Flight function
def levy_flight(size):
    return np.sum(np.tan(math.pi * np.random.uniform(low=0, high=1, size=(1, size))))

def migration_operator(migrant_male_swarm_pos, female_swarm_pos, male_swarm_pos, peri, p):
    D = len(migrant_male_swarm_pos[0])  # Assuming all butterflies have the same dimensionality D

    for i in range(subpop_size):
        # Evaluate fitness for the current butterfly
        current_fitness = find_fitness(migrant_male_swarm_pos[i][1:])

        for k in range(1, D):  # Starting from 1 as the 0th element is skipped (fitness)
            rand = np.random.uniform(low=0, high=1)
            r = rand * peri
            if r <= p:
                random_female_index = np.random.randint(0, subpop_size)
                selected_butterfly = female_swarm_pos[random_female_index]
            else:
                random_female_index = np.random.randint(0, subpop_size)
```

```
                selected_butterfly = male_swarm_pos[random_female_index]

            # Generate the kth element of the new butterfly
            migrant_male_swarm_pos[i][k] = selected_butterfly[k]

        # Evaluate fitness for the new butterfly
        new_fitness = find_fitness(migrant_male_swarm_pos[i][1:])

        # If the new fitness is better, update the 0th element of the butterfly
        if new_fitness < current_fitness:
            migrant_male_swarm_pos[i][0] = new_fitness

        # Print relevant information for debugging
        #print(f"For mayfly {i}: Current Fitness: {current_fitness}, New Fitness: {new_fitness}")

    return migrant_male_swarm_pos

def adjusting_operator(female_swarm_pos, male_swarm_pos, t):
    t += 1
    for i in range(subpop_size):
        if np.random.rand() < adjusting_rate:
            rand = np.random.uniform(low=0, high=1)
            if rand <= p:
                # Best butterfly will be the first, irrespective of NP designation
                if female_swarm_pos[0][0] <= male_swarm_pos[0][0]:
                    selected_butterfly = female_swarm_pos[0]
                else:
                    selected_butterfly = male_swarm_pos[0]

                levy_factor = s_max / (t**2)
                # Perform Levy flight perturbation on the selected butterfly
                for k in range(1, len(selected_butterfly)):
                    selected_butterfly[k] += levy_factor * (levy_flight(lf_size) - 0.5)
                    #selected_butterfly[k] += levy_factor * (levy_flight(selected_butterfly[k], lf_size) - 0.5)

                # Check if the selected butterfly has better fitness
                selected_fitness = find_fitness(selected_butterfly[1:])
                current_fitness = find_fitness(female_swarm_pos[i][1:])

                if selected_fitness < current_fitness:
                    # Replace a butterfly in the current population with the selected one
                    female_swarm_pos[i] = selected_butterfly.copy()
            else:
                # Randomly select a butterfly from the opposite gender
                random_male_or_female_index = np.random.randint(0, subpop_size)
                selected_butterfly = male_swarm_pos[random_male_or_female_index]

                # Check if the selected butterfly has better fitness
                selected_fitness = find_fitness(selected_butterfly[1:])
                current_fitness = find_fitness(female_swarm_pos[i][1:])

                if selected_fitness < current_fitness:
                    # Replace a butterfly in the current population with the selected one
                    female_swarm_pos[i] = selected_butterfly.copy()

    return female_swarm_pos

def update_vmax(swarm_pos_male, swarm_pos_female, vmax_male, vmax_female, subpop_size):
    for j in range(len(vmax_male)):
        r = np.random.normal(0, 1)
        index_male = min(subpop_size - 1, len(swarm_pos_male) - 1)
        index_female = min(subpop_size - 1, len(swarm_pos_female) - 1)

        vmax_male[j] = (swarm_pos_male[0][j] - swarm_pos_male[index_male][j]) * r
        vmax_female[j] = (swarm_pos_female[0][j] - swarm_pos_female[index_female][j]) * r

def sort_population(population_pos, population_vel):
    # Calculate fitness for each individual
    population_fitness = np.array([find_fitness(individual) for individual in population_pos])

    # Sort the population based on fitness
```

```python
    sort_order = np.argsort(population_fitness)
    population_fitness = population_fitness[sort_order]
    population_pos = population_pos[sort_order]
    population_vel = population_vel[sort_order]

    return population_fitness, population_pos, population_vel

def update_gbest(gbest_fitness, gbest, male_fitness, male_swarm_pos, female_fitness, female_swarm_pos):
    if male_fitness[0] < gbest_fitness:
        gbest_fitness = male_fitness[0]
        gbest = male_swarm_pos[0].copy()

    if female_fitness[0] < gbest_fitness:
        gbest_fitness = female_fitness[0]
        gbest = female_swarm_pos[0].copy()

    return gbest_fitness, gbest

def crossover_and_mutation(NP_male_swarm_pos, NP_female_swarm_pos, NP_male_swarm_vel, NP_female_swarm_vel, tot_features,
 subpop_size, l):
    NP_offspring1 = np.zeros((subpop_size, tot_features))
    NP_offspring2 = np.zeros((subpop_size, tot_features))

    for i in range(subpop_size):
        # Crossover
        partition = np.random.randint(tot_features // 4, math.floor((3 * tot_features // 4) + 1))
        for j in range(tot_features):
            NP_offspring1[i][j] = l * NP_male_swarm_pos[i][j] + (1 - l) * NP_female_swarm_pos[i][j]
            NP_offspring2[i][j] = l * NP_female_swarm_pos[i][j] + (1 - l) * NP_male_swarm_pos[i][j]

        if np.random.random() >= 0.5:
            NP_male_swarm_pos[i] = NP_offspring1[i].copy()
            NP_female_swarm_pos[i] = NP_offspring2[i].copy()
        else:
            NP_male_swarm_pos[i] = NP_offspring2[i].copy()
            NP_female_swarm_pos[i] = NP_offspring1[i].copy()

        # Mutation
        mutation_strength = 1.2
        r = np.random.exponential(scale=mutation_strength, size=tot_features)
        for j in range(tot_features):
            NP_male_swarm_pos[i][j] += r[j]
            NP_female_swarm_pos[i][j] += r[j]

        # Reset velocities
        NP_male_swarm_vel[i] = np.zeros(tot_features)
        NP_female_swarm_vel[i] = np.zeros(tot_features)

    return NP_male_swarm_pos, NP_female_swarm_pos, NP_male_swarm_vel, NP_female_swarm_vel

def update_swarm_position(swarm_pos, swarm_vel):
    for i in range(len(swarm_pos)):
        for j in range(len(swarm_pos[i])):
            swarm_pos[i][j] += swarm_vel[i][j]

def elitism(population_male_swarm_pos, population_male_swarm_vel, population_female_swarm_pos, population_female_swarm_v
el, elite_swarm_pos, elite_swarm_fitness, num_elite, tot_features):
    # Update Elite Individuals
    if find_fitness(population_male_swarm_pos[0]) < elite_swarm_fitness[0]:
        elite_swarm_fitness[0] = find_fitness(population_male_swarm_pos[0])
        elite_swarm_pos[0] = population_male_swarm_pos[0].copy()

    if find_fitness(population_female_swarm_pos[0]) < elite_swarm_fitness[0]:
        elite_swarm_fitness[0] = find_fitness(population_female_swarm_pos[0])
        elite_swarm_pos[0] = population_female_swarm_pos[0].copy()

    # Replace the worst individuals with the elite individuals
    for i in range(num_elite):
        population_male_swarm_pos[-1-i] = elite_swarm_pos[i].copy()
        population_male_swarm_vel[-1-i] = np.zeros(tot_features)
        population_female_swarm_pos[-1-i] = elite_swarm_pos[i].copy()
        population_female_swarm_vel[-1-i] = np.zeros(tot_features)
```

```python
# Sorting initially for NP1 males
NP1_male_fitness, NP1_male_swarm_pos, NP1_male_swarm_vel = sort_population(NP1_male_swarm_pos, NP1_male_swarm_vel)
# Sorting initially for NP1 females
NP1_female_fitness, NP1_female_swarm_pos, NP1_female_swarm_vel = sort_population(NP1_female_swarm_pos, NP1_female_swarm_
vel)
# Sorting initially for NP2 males
NP2_male_fitness, NP2_male_swarm_pos, NP2_male_swarm_vel = sort_population(NP2_male_swarm_pos, NP2_male_swarm_vel)
# Sorting initially for NP2 females
NP2_female_fitness, NP2_female_swarm_pos, NP2_female_swarm_vel = sort_population(NP2_female_swarm_pos, NP2_female_swarm_
vel)

# Gbest updation for NP1 and NP2
gbest_fitness, gbest = update_gbest(gbest_fitness, gbest, NP1_male_fitness, NP1_male_swarm_pos, NP1_female_fitness, NP1_
female_swarm_pos)
gbest_fitness, gbest = update_gbest(gbest_fitness, gbest, NP2_male_fitness, NP2_male_swarm_pos, NP2_female_fitness, NP2_
female_swarm_pos)

# Add a variable to store the elite individuals
num_elite = 1
elite_swarm_pos = np.zeros((num_elite, tot_features))
elite_swarm_fitness = np.empty(num_elite)
elite_swarm_fitness.fill(np.inf)

# Initialize an empty list to store the convergence curve
# mmo_convergence_curve = []

# Main Monarch Mayfly Optimization
for itr in range(max_iterations):
    # Updating vmax (velocity limit) for NP1
    update_vmax(NP1_male_swarm_pos, NP1_female_swarm_pos, NP1_vmax_male, NP1_vmax_female, subpop_size)

    # Updating vmax (velocity limit) for NP2
    update_vmax(NP2_male_swarm_pos, NP2_female_swarm_pos, NP2_vmax_male, NP2_vmax_female, subpop_size)

    ####################################################### NP1 #######################################################

    # NP1 Males
    for i in range(subpop_size):
        if NP1_male_fitness[i] < gbest_fitness:
            gbest_fitness = NP1_male_fitness[i]
            gbest = NP1_male_swarm_pos[i].copy()

        if NP1_male_fitness[i] < pbest_fitness[i]:
            pbest_fitness[i] = NP1_male_fitness[i]
            pbest[i] = NP1_male_swarm_pos[i].copy()

        # Velocity updation for NP1 males
        if i == 0:
            for j in range(tot_features):
                NP1_male_swarm_vel[0][j] = NP1_male_swarm_vel[0][j] + d * np.random.uniform(-1, 1)
        else:
            sum = 0
            for j in range(tot_features):
                sum = sum + (NP1_male_swarm_pos[i][j] - gbest[j]) * (NP1_male_swarm_pos[i][j] - gbest[j])
            rg = math.sqrt(sum)
            sum = 0
            for j in range(tot_features):
                sum = sum + (NP1_male_swarm_pos[i][j] - pbest[i][j]) * (NP1_male_swarm_pos[i][j] - pbest[i][j])
            rp = math.sqrt(sum)
            for j in range(tot_features):
                NP1_male_swarm_vel[i][j] = g * NP1_male_swarm_vel[i][j] + a1 * math.exp(-beta * rp * rp) * (
                        pbest[i][j] - NP1_male_swarm_pos[i][j]) + a2 * math.exp(-beta * rg * rg) * (
                                                   gbest[j] - NP1_male_swarm_pos[i][j])
                # Include attraction towards personal best (pbest)
                NP1_male_swarm_vel[i][j] += a1 * (pbest[i][j] - NP1_male_swarm_pos[i][j])

                # Include attraction towards global best (gbest)
                NP1_male_swarm_vel[i][j] += a2 * (gbest[j] - NP1_male_swarm_pos[i][j])


        # Velocity updation for NP1 females
        if NP1_female_fitness[i] <= NP1_male_fitness[i]:
```

```
            sum = 0
            for j in range(tot_features):
                sum = sum + (NP1_male_swarm_pos[i][j] - NP1_female_swarm_pos[i][j]) * (NP1_male_swarm_pos[i][j] - NP1_fe
male_swarm_pos[i][j])
            rmf = math.sqrt(sum)
            NP1_female_swarm_vel[i][j] = g * NP1_female_swarm_vel[i][j] + a2 * math.exp(-beta * rmf * rmf) * (NP1_male_s
warm_pos[i][j] - NP1_female_swarm_pos[i][j])
        else:
            for j in range(tot_features):
                NP1_female_swarm_vel[i][j] = g * NP1_female_swarm_vel[i][j] + fl * np.random.uniform(-1, 1)


    # Levy flight for NP1 females
    for i in range(subpop_size):
        if NP1_female_fitness[i] > NP1_male_fitness[i]:
            # Employ Levy flight for exploration
            for j in range(tot_features):
                NP1_female_swarm_pos[i][j] += levy_flight(lf_size)
        else:
            # Perform...
            for j in range(tot_features):
                NP1_female_swarm_pos[i][j] = np.random.uniform(-1, 1)


    # Call migration operator for NP1 males
    NP1_male_swarm_pos = migration_operator(NP1_male_swarm_pos, NP1_female_swarm_pos, NP2_male_swarm_pos, 1.2, float(5/1
2))
    # Call adjusting operator for NP2 females
    NP1_female_swarm_pos = adjusting_operator(NP1_female_swarm_pos, NP2_male_swarm_pos, itr)


    # Sorting for NP1 males
    NP1_male_fitness, NP1_male_swarm_pos, NP1_male_swarm_vel = sort_population(NP1_male_swarm_pos, NP1_male_swarm_vel)
    # Sorting for NP1 females
    NP1_female_fitness, NP1_female_swarm_pos, NP1_female_swarm_vel = sort_population(NP1_female_swarm_pos, NP1_female_sw
arm_vel)

    # Gbest updation
    gbest_fitness, gbest = update_gbest(gbest_fitness, gbest, NP1_male_fitness, NP1_male_swarm_pos, NP1_female_fitness,
NP1_female_swarm_pos)

    # Crossover and mutation for NP1
    NP1_male_swarm_pos, NP1_female_swarm_pos, NP1_male_swarm_vel, NP1_female_swarm_vel = crossover_and_mutation(NP1_male
_swarm_pos, NP1_female_swarm_pos, NP1_male_swarm_vel, NP1_female_swarm_vel, tot_features, subpop_size, l)

    # Updating swarm position for NP1
    update_swarm_position(NP1_male_swarm_pos, NP1_male_swarm_vel)
    update_swarm_position(NP1_female_swarm_pos, NP1_female_swarm_vel)

    # ELITISM: Save the elite individuals and Replace the worst individuals with the elite individuals for NP1
    elitism(NP1_male_swarm_pos, NP1_male_swarm_vel, NP1_female_swarm_pos, NP1_female_swarm_vel, elite_swarm_pos, elite_s
warm_fitness, num_elite, tot_features)

    # Updating gravity and nuptial dance for NP1
    d = d * delta
    fl = fl * delta

    # Gbest updation
    gbest_fitness, gbest = update_gbest(gbest_fitness, gbest, NP1_male_fitness, NP1_male_swarm_pos, NP1_female_fitness,
NP1_female_swarm_pos)

    ####################################################### NP2 #######################################################

    # NP2 Males
    iterations_without_improvement = 0
    while iterations_without_improvement < max_neighbors:
        for i in range(subpop_size):
            if NP2_male_fitness[i] < gbest_fitness:
                gbest_fitness = NP2_male_fitness[i]
                gbest = NP2_male_swarm_pos[i].copy()
```

```
            if NP2_male_fitness[i] < pbest_fitness[i]:
                pbest_fitness[i] = NP2_male_fitness[i]
                pbest[i] = NP2_male_swarm_pos[i].copy()

            # Velocity updation for NP2 males
            if i == 0:
                for j in range(tot_features):
                    NP2_male_swarm_vel[0][j] = NP2_male_swarm_vel[0][j] + d * np.random.uniform(-1, 1)
            else:
                sum = 0
                for j in range(tot_features):
                    sum = sum + (NP2_male_swarm_pos[i][j] - gbest[j]) * (NP2_male_swarm_pos[i][j] - gbest[j])
                rg = math.sqrt(sum)
                sum = 0
                for j in range(tot_features):
                    sum = sum + (NP2_male_swarm_pos[i][j] - pbest[i][j]) * (NP2_male_swarm_pos[i][j] - pbest[i][j])
                rp = math.sqrt(sum)
                for j in range(tot_features):
                    NP2_male_swarm_vel[i][j] = g * NP2_male_swarm_vel[i][j] + a1 * math.exp(-beta * rp * rp) * (
                            pbest[i][j] - NP2_male_swarm_pos[i][j]) + a2 * math.exp(-beta * rg * rg) * (
                                                       gbest[j] - NP2_male_swarm_pos[i][j])
                    # Include attraction towards personal best (pbest)
                    NP2_male_swarm_vel[i][j] += a1 * (pbest[i][j] - NP2_male_swarm_pos[i][j])

                    # Include attraction towards global best (gbest)
                    NP2_male_swarm_vel[i][j] += a2 * (gbest[j] - NP2_male_swarm_pos[i][j])

            # Velocity updation for NP2 females
            if NP2_female_fitness[i] <= NP2_male_fitness[i]:
                sum = 0
                for j in range(tot_features):
                    sum = sum + (NP2_male_swarm_pos[i][j] - NP2_female_swarm_pos[i][j]) * (NP2_male_swarm_pos[i][j] - NP
2_female_swarm_pos[i][j])
                rmf = math.sqrt(sum)
                NP2_female_swarm_vel[i][j] = g * NP2_female_swarm_vel[i][j] + a2 * math.exp(-beta * rmf * rmf) * (NP2_ma
le_swarm_pos[i][j] - NP2_female_swarm_pos[i][j])
            else:
                for j in range(tot_features):
                    NP2_female_swarm_vel[i][j] = g * NP2_female_swarm_vel[i][j] + fl * np.random.uniform(-1, 1)

        # Levy flight for NP2 females
        for i in range(subpop_size):
            if NP2_female_fitness[i] > NP2_male_fitness[i]:
                # Employ Levy flight for exploration
                for j in range(tot_features):
                    NP2_female_swarm_pos[i][j] += levy_flight(lf_size)
            else:
                # Perform...
                for j in range(tot_features):
                    NP2_female_swarm_pos[i][j] = np.random.uniform(-1, 1)

        # Call migration operator for NP2 males
        NP2_male_swarm_pos = migration_operator(NP2_male_swarm_pos, NP2_female_swarm_pos, NP1_male_swarm_pos, 1.2, float
(5/12))
        # Call adjusting operator for NP2 females
        NP2_female_swarm_pos = adjusting_operator(NP2_female_swarm_pos, NP1_male_swarm_pos, itr)

        # Sorting and updating for NP2 males
        NP2_male_fitness, NP2_male_swarm_pos, NP2_male_swarm_vel = sort_population(NP2_male_swarm_pos, NP2_male_swarm_ve
l)
        # Sorting and updating for NP2 females
        NP2_female_fitness, NP2_female_swarm_pos, NP2_female_swarm_vel = sort_population(NP2_female_swarm_pos, NP2_femal
e_swarm_vel)

        #Gbest updation for NP2
        new_gbest_fitness, new_gbest = update_gbest(gbest_fitness, gbest, NP2_male_fitness, NP2_male_swarm_pos, NP2_fema
le_fitness, NP2_female_swarm_pos)

        ############ v2 ###############################
        if new_gbest_fitness < gbest_fitness:
            gbest_fitness = new_gbest_fitness
            gbest = new_gbest.copy()
```

```
            print("Iteration:", itr, "GBest Fitness:", gbest_fitness)
            iterations_without_improvement = 0  # Reset the counter when a better solution is found
        else:
            print("Iteration:", itr, "New GBest Fitness:", new_gbest_fitness)
            iterations_without_improvement += 1  # Increment the counter when no improvement is found

        # Crossover and mutation for NP2
        NP2_male_swarm_pos, NP2_female_swarm_pos, NP2_male_swarm_vel, NP2_female_swarm_vel = crossover_and_mutation(NP2_
male_swarm_pos, NP2_female_swarm_pos, NP2_male_swarm_vel, NP2_female_swarm_vel, tot_features, subpop_size, l)

        # Updating swarm position for NP2
        update_swarm_position(NP2_male_swarm_pos, NP2_male_swarm_vel)
        update_swarm_position(NP2_female_swarm_pos, NP2_female_swarm_vel)

        # ELITISM: Save the elite individuals and Replace the worst individuals with the elite individuals for NP2
        elitism(NP2_male_swarm_pos, NP2_male_swarm_vel, NP2_female_swarm_pos, NP2_female_swarm_vel, elite_swarm_pos, eli
te_swarm_fitness, num_elite, tot_features)

        # Updating gravity and nuptial dance for NP2
        g = gmax-((gmax-gmin)*itr/max_iterations)
        d = d * delta
        fl = fl * delta

        #Gbest updation for NP2
        gbest_fitness, gbest = update_gbest(gbest_fitness, gbest, NP2_male_fitness, NP2_male_swarm_pos, NP2_female_fitne
ss, NP2_female_swarm_pos)

    # Append the current gbest_fitness to the convergence_curve list
    # mmo_convergence_curve.append(gbest_fitness)
    print("Iteration:", itr, "GBest Fitness:", gbest_fitness)

# Gbest = transfer_func(gbest)
selected_features = transfer_func(gbest)
for j in range(tot_features):
    if selected_features[j] >= 0.25:
        selected_features[j] = 1
    else:
        selected_features[j] = 0
number_of_selected_features = np.sum(selected_features)
print("NUM:", number_of_selected_features)
print("SELECTED FEATURES:", selected_features)

features = [df.columns[j] for j in range(len(selected_features)) if selected_features[j] == 1]
if not features:
    acc = 0
else:
    new_x_train = x_train[features]
    new_x_test = x_test[features]
    _classifier = KNeighborsClassifier(n_neighbors=5)
    _classifier.fit(new_x_train, y_train)
    predictions = _classifier.predict(new_x_test)
    acc = accuracy_score(y_true=y_test, y_pred=predictions)
    fitness = acc
print("ACC:", acc)

end_time=process_time()
print("time in seconds =",(end_time-start_time))
print("time in microseconds =",(end_time-start_time) * 1e6)
```

APPENDIX B

MMO-SVM SOURCE CODE

```
"""
Monarch Mayfly Optimization (MMO) Algorithm for Feature Selection
=================================================================
This implementation of the Monarch Mayfly Optimization (MMO) algorithm
is a hybrid of the Monarch Butterfly Optimization (MBO) and the Mayfly Algorithm (MA).

Sources and Acknowledgements:
-----------------------------
1. Monarch Butterfly Optimization (MBO):
   - Original implementation in Python by Justin van Zyl.
   - Based on the study:
     Wang G., Deb S., Cui Z., "Monarch Butterfly Optimization," Neural Comput & Applic 31:1995-2014.
     doi: 10.1007/s00521-015-1923-y.
   - Key operations utilized: Migration Operator, Adjusting Operator, and Elitism.

2. Mayfly Algorithm (MA):
   - Extracted from the hybrid feature selection study:
     Bhattacharyya, T., Chatterjee, B., Singh, P. K., Yoon, J. H., Geem, Z. W., & Sarkar, R. (2020).
     "Mayfly in harmony: A new hybrid meta-heuristic feature selection algorithm," IEEE Access, 8, 195929-195945.
   - The study hybridized MA with Harmony Search (HS). The MA part is used in this MMO hybrid.

Disclaimer:
-----------
This code is a hybrid implementation of the aforementioned algorithms and combines elements from both
to create the MMO algorithm for the purpose of feature selection. Full credit goes to the original authors
for their contributions.
"""
import numpy as np
import pandas as pd
import math
import random
from statistics import stdev
from time import process_time
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import matplotlib.pyplot as plt

start_time = process_time()

# Load dataset
df = pd.read_csv('Breastcancer.csv')
tot_features = len(df.columns) - 1
x = df[df.columns[:tot_features]]
y = df[df.columns[-1]]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stratify=y)

# Train classifier using original dataset
_classifier = SVC()
_classifier.fit(x_train, y_train)
predictions = _classifier.predict(x_test)
total_acc = accuracy_score(y_true=y_test, y_pred=predictions)
total_error = 1 - total_acc
total_features = tot_features
total_acc

# Controlling parameters
swarm_size = 20
max_iterations = 20
alpha = 0.01
a1 = 3
a2 = 3.5
beta = 0.1
d = 3
fl = 3
l = 0.95
g = 1
delta = 0.9
lf_size = 1
```

```python
adjusting_rate = 0.1
p = float(6/12)
s_max = 0.02
gmax=9.8
gmin=6
max_neighbors = 20

# Population structure and Initialization
subpop_size = swarm_size // 2
NP1_male_swarm_vel = np.zeros((subpop_size, tot_features))
NP1_female_swarm_vel = np.zeros((subpop_size, tot_features))
NP2_male_swarm_vel = np.zeros((subpop_size, tot_features))
NP2_female_swarm_vel = np.zeros((subpop_size, tot_features))

NP1_male_swarm_pos = np.random.uniform(low=-1, high=1, size=(subpop_size, tot_features))
NP1_female_swarm_pos = np.random.uniform(low=-1, high=1, size=(subpop_size, tot_features))
NP2_male_swarm_pos = np.random.uniform(low=-1, high=1, size=(subpop_size, tot_features))
NP2_female_swarm_pos = np.random.uniform(low=-1, high=1, size=(subpop_size, tot_features))

gbest_fitness = 1000000
pbest_fitness = np.empty(swarm_size)
pbest_fitness.fill(np.inf)
pbest = np.zeros((swarm_size, tot_features))
gbest = np.zeros(tot_features)
NP1_male_fitness = np.empty(subpop_size)
NP1_female_fitness = np.empty(subpop_size)
NP2_male_fitness = np.empty(subpop_size)
NP2_female_fitness = np.empty(subpop_size)
NP1_vmax_male = np.empty(tot_features)
NP1_vmax_female = np.empty(tot_features)
NP2_vmax_male = np.empty(tot_features)
NP2_vmax_female = np.empty(tot_features)

# S-shaped transfer function
def transfer_func(velocity):
    s1 = np.abs(velocity) * 0.5 + 1
    s1 = (-velocity) / s1 + 0.5
    return s1

# Fitness function
def find_fitness(particle):
    features = [df.columns[i] for i, v in enumerate(transfer_func(particle)) if v >= 0.25]
    if not features:
        return 10000
    new_x_train = x_train[features].copy()
    new_x_test = x_test[features].copy()
    _classifier = SVC()
    _classifier.fit(new_x_train, y_train)
    predictions = _classifier.predict(new_x_test)
    acc = accuracy_score(y_true=y_test, y_pred=predictions)
    err = 1 - acc
    num_features = len(features)
    fitness = alpha * (num_features / total_features) + (1 - alpha) * err
    return fitness

# Levy Flight function
def levy_flight(size):
    return np.sum(np.tan(math.pi * np.random.uniform(low=0, high=1, size=(1, size))))

def migration_operator(migrant_male_swarm_pos, female_swarm_pos, male_swarm_pos, peri, p):
    D = len(migrant_male_swarm_pos[0])  # Assuming all butterflies have the same dimensionality D

    for i in range(subpop_size):
        # Evaluate fitness for the current butterfly
        current_fitness = find_fitness(migrant_male_swarm_pos[i][1:])

        for k in range(1, D):  # Starting from 1 as the 0th element is skipped (fitness)
            rand = np.random.uniform(low=0, high=1)
            r = rand * peri
            if r <= p:
                random_female_index = np.random.randint(0, subpop_size)
                selected_butterfly = female_swarm_pos[random_female_index]
```

```python
            else:
                random_female_index = np.random.randint(0, subpop_size)
                selected_butterfly = male_swarm_pos[random_female_index]

            # Generate the kth element of the new butterfly
            migrant_male_swarm_pos[i][k] = selected_butterfly[k]

        # Evaluate fitness for the new butterfly
        new_fitness = find_fitness(migrant_male_swarm_pos[i][1:])

        # If the new fitness is better, update the 0th element of the butterfly
        if new_fitness < current_fitness:
            migrant_male_swarm_pos[i][0] = new_fitness

        # Print relevant information for debugging
        #print(f"For mayfly {i}: Current Fitness: {current_fitness}, New Fitness: {new_fitness}")

    return migrant_male_swarm_pos

def adjusting_operator(female_swarm_pos, male_swarm_pos, t):
    t += 1
    for i in range(subpop_size):
        if np.random.rand() < adjusting_rate:
            rand = np.random.uniform(low=0, high=1)
            if rand <= p:
                # Best butterfly will be the first, irrespective of NP designation
                if female_swarm_pos[0][0] <= male_swarm_pos[0][0]:
                    selected_butterfly = female_swarm_pos[0]
                else:
                    selected_butterfly = male_swarm_pos[0]

                levy_factor = s_max / (t**2)
                # Perform Levy flight perturbation on the selected butterfly
                for k in range(1, len(selected_butterfly)):
                    selected_butterfly[k] += levy_factor * (levy_flight(lf_size) - 0.5)
                    #selected_butterfly[k] += levy_factor * (levy_flight(selected_butterfly[k], lf_size) - 0.5)

                # Check if the selected butterfly has better fitness
                selected_fitness = find_fitness(selected_butterfly[1:])
                current_fitness = find_fitness(female_swarm_pos[i][1:])

                if selected_fitness < current_fitness:
                    # Replace a butterfly in the current population with the selected one
                    female_swarm_pos[i] = selected_butterfly.copy()
            else:
                # Randomly select a butterfly from the opposite gender
                random_male_or_female_index = np.random.randint(0, subpop_size)
                selected_butterfly = male_swarm_pos[random_male_or_female_index]

                # Check if the selected butterfly has better fitness
                selected_fitness = find_fitness(selected_butterfly[1:])
                current_fitness = find_fitness(female_swarm_pos[i][1:])

                if selected_fitness < current_fitness:
                    # Replace a butterfly in the current population with the selected one
                    female_swarm_pos[i] = selected_butterfly.copy()

    return female_swarm_pos

def update_vmax(swarm_pos_male, swarm_pos_female, vmax_male, vmax_female, subpop_size):
    for j in range(len(vmax_male)):
        r = np.random.normal(0, 1)
        index_male = min(subpop_size - 1, len(swarm_pos_male) - 1)
        index_female = min(subpop_size - 1, len(swarm_pos_female) - 1)

        vmax_male[j] = (swarm_pos_male[0][j] - swarm_pos_male[index_male][j]) * r
        vmax_female[j] = (swarm_pos_female[0][j] - swarm_pos_female[index_female][j]) * r

def sort_population(population_pos, population_vel):
    # Calculate fitness for each individual
    population_fitness = np.array([find_fitness(individual) for individual in population_pos])
```

```python
    # Sort the population based on fitness
    sort_order = np.argsort(population_fitness)
    population_fitness = population_fitness[sort_order]
    population_pos = population_pos[sort_order]
    population_vel = population_vel[sort_order]

    return population_fitness, population_pos, population_vel

def update_gbest(gbest_fitness, gbest, male_fitness, male_swarm_pos, female_fitness, female_swarm_pos):
    if male_fitness[0] < gbest_fitness:
        gbest_fitness = male_fitness[0]
        gbest = male_swarm_pos[0].copy()

    if female_fitness[0] < gbest_fitness:
        gbest_fitness = female_fitness[0]
        gbest = female_swarm_pos[0].copy()

    return gbest_fitness, gbest

def crossover_and_mutation(NP_male_swarm_pos, NP_female_swarm_pos, NP_male_swarm_vel, NP_female_swarm_vel, tot_features,
 subpop_size, l):
    NP_offspring1 = np.zeros((subpop_size, tot_features))
    NP_offspring2 = np.zeros((subpop_size, tot_features))

    for i in range(subpop_size):
        # Crossover
        partition = np.random.randint(tot_features // 4, math.floor((3 * tot_features // 4) + 1))
        for j in range(tot_features):
            NP_offspring1[i][j] = l * NP_male_swarm_pos[i][j] + (1 - l) * NP_female_swarm_pos[i][j]
            NP_offspring2[i][j] = l * NP_female_swarm_pos[i][j] + (1 - l) * NP_male_swarm_pos[i][j]

        if np.random.random() >= 0.5:
            NP_male_swarm_pos[i] = NP_offspring1[i].copy()
            NP_female_swarm_pos[i] = NP_offspring2[i].copy()
        else:
            NP_male_swarm_pos[i] = NP_offspring2[i].copy()
            NP_female_swarm_pos[i] = NP_offspring1[i].copy()

        # Mutation
        mutation_strength = 1.2
        r = np.random.exponential(scale=mutation_strength, size=tot_features)
        for j in range(tot_features):
            NP_male_swarm_pos[i][j] += r[j]
            NP_female_swarm_pos[i][j] += r[j]

        # Reset velocities
        NP_male_swarm_vel[i] = np.zeros(tot_features)
        NP_female_swarm_vel[i] = np.zeros(tot_features)

    return NP_male_swarm_pos, NP_female_swarm_pos, NP_male_swarm_vel, NP_female_swarm_vel

def update_swarm_position(swarm_pos, swarm_vel):
    for i in range(len(swarm_pos)):
        for j in range(len(swarm_pos[i])):
            swarm_pos[i][j] += swarm_vel[i][j]

def elitism(population_male_swarm_pos, population_male_swarm_vel, population_female_swarm_pos, population_female_swarm_v
el, elite_swarm_pos, elite_swarm_fitness, num_elite, tot_features):
    # Update Elite Individuals
    if find_fitness(population_male_swarm_pos[0]) < elite_swarm_fitness[0]:
        elite_swarm_fitness[0] = find_fitness(population_male_swarm_pos[0])
        elite_swarm_pos[0] = population_male_swarm_pos[0].copy()

    if find_fitness(population_female_swarm_pos[0]) < elite_swarm_fitness[0]:
        elite_swarm_fitness[0] = find_fitness(population_female_swarm_pos[0])
        elite_swarm_pos[0] = population_female_swarm_pos[0].copy()

    # Replace the worst individuals with the elite individuals
    for i in range(num_elite):
        population_male_swarm_pos[-1-i] = elite_swarm_pos[i].copy()
```

```python
        population_male_swarm_vel[-1-i] = np.zeros(tot_features)
        population_female_swarm_pos[-1-i] = elite_swarm_pos[i].copy()
        population_female_swarm_vel[-1-i] = np.zeros(tot_features)

# Sorting initially for NP1 males
NP1_male_fitness, NP1_male_swarm_pos, NP1_male_swarm_vel = sort_population(NP1_male_swarm_pos, NP1_male_swarm_vel)
# Sorting initially for NP1 females
NP1_female_fitness, NP1_female_swarm_pos, NP1_female_swarm_vel = sort_population(NP1_female_swarm_pos, NP1_female_swarm_
vel)
# Sorting initially for NP2 males
NP2_male_fitness, NP2_male_swarm_pos, NP2_male_swarm_vel = sort_population(NP2_male_swarm_pos, NP2_male_swarm_vel)
# Sorting initially for NP2 females
NP2_female_fitness, NP2_female_swarm_pos, NP2_female_swarm_vel = sort_population(NP2_female_swarm_pos, NP2_female_swarm_
vel)

# Gbest updation for NP1 and NP2
gbest_fitness, gbest = update_gbest(gbest_fitness, gbest, NP1_male_fitness, NP1_male_swarm_pos, NP1_female_fitness, NP1_
female_swarm_pos)
gbest_fitness, gbest = update_gbest(gbest_fitness, gbest, NP2_male_fitness, NP2_male_swarm_pos, NP2_female_fitness, NP2_
female_swarm_pos)

# Add a variable to store the elite individuals
num_elite = 1
elite_swarm_pos = np.zeros((num_elite, tot_features))
elite_swarm_fitness = np.empty(num_elite)
elite_swarm_fitness.fill(np.inf)

# Initialize an empty list to store the convergence curve
convergence_curve = []

# Main Monarch Mayfly Optimization
for itr in range(max_iterations):
    # Updating vmax (velocity limit) for NP1
    update_vmax(NP1_male_swarm_pos, NP1_female_swarm_pos, NP1_vmax_male, NP1_vmax_female, subpop_size)

    # Updating vmax (velocity limit) for NP2
    update_vmax(NP2_male_swarm_pos, NP2_female_swarm_pos, NP2_vmax_male, NP2_vmax_female, subpop_size)

    ####################################################### NP1 ########################################################

    # NP1 Males
    for i in range(subpop_size):
        if NP1_male_fitness[i] < gbest_fitness:
            gbest_fitness = NP1_male_fitness[i]
            gbest = NP1_male_swarm_pos[i].copy()

        if NP1_male_fitness[i] < pbest_fitness[i]:
            pbest_fitness[i] = NP1_male_fitness[i]
            pbest[i] = NP1_male_swarm_pos[i].copy()

        # Velocity updation for NP1 males
        if i == 0:
            for j in range(tot_features):
                NP1_male_swarm_vel[0][j] = NP1_male_swarm_vel[0][j] + d * np.random.uniform(-1, 1)
        else:
            sum = 0
            for j in range(tot_features):
                sum = sum + (NP1_male_swarm_pos[i][j] - gbest[j]) * (NP1_male_swarm_pos[i][j] - gbest[j])
            rg = math.sqrt(sum)
            sum = 0
            for j in range(tot_features):
                sum = sum + (NP1_male_swarm_pos[i][j] - pbest[i][j]) * (NP1_male_swarm_pos[i][j] - pbest[i][j])
            rp = math.sqrt(sum)
            for j in range(tot_features):
                NP1_male_swarm_vel[i][j] = g * NP1_male_swarm_vel[i][j] + a1 * math.exp(-beta * rp * rp) * (
                        pbest[i][j] - NP1_male_swarm_pos[i][j]) + a2 * math.exp(-beta * rg * rg) * (
                                          gbest[j] - NP1_male_swarm_pos[i][j])
                # Include attraction towards personal best (pbest)
                NP1_male_swarm_vel[i][j] += a1 * (pbest[i][j] - NP1_male_swarm_pos[i][j])

                # Include attraction towards global best (gbest)
                NP1_male_swarm_vel[i][j] += a2 * (gbest[j] - NP1_male_swarm_pos[i][j])
```

```python
        # Velocity updation for NP1 females
        if NP1_female_fitness[i] <= NP1_male_fitness[i]:
            sum = 0
            for j in range(tot_features):
                sum = sum + (NP1_male_swarm_pos[i][j] - NP1_female_swarm_pos[i][j]) * (NP1_male_swarm_pos[i][j] - NP1_fe
male_swarm_pos[i][j])
            rmf = math.sqrt(sum)
            NP1_female_swarm_vel[i][j] = g * NP1_female_swarm_vel[i][j] + a2 * math.exp(-beta * rmf * rmf) * (NP1_male_s
warm_pos[i][j] - NP1_female_swarm_pos[i][j])
        else:
            for j in range(tot_features):
                NP1_female_swarm_vel[i][j] = g * NP1_female_swarm_vel[i][j] + fl * np.random.uniform(-1, 1)


    # Levy flight for NP1 females
    for i in range(subpop_size):
        if NP1_female_fitness[i] > NP1_male_fitness[i]:
            # Employ Levy flight for exploration
            for j in range(tot_features):
                NP1_female_swarm_pos[i][j] += levy_flight(lf_size)
        else:
            # Perform...
            for j in range(tot_features):
                NP1_female_swarm_pos[i][j] = np.random.uniform(-1, 1)



    # Call migration operator for NP1 males
    NP1_male_swarm_pos = migration_operator(NP1_male_swarm_pos, NP1_female_swarm_pos, NP2_male_swarm_pos, 1.2, float(5/1
2))
    # Call adjusting operator for NP2 females
    NP1_female_swarm_pos = adjusting_operator(NP1_female_swarm_pos, NP2_male_swarm_pos, itr)



    # Sorting for NP1 males
    NP1_male_fitness, NP1_male_swarm_pos, NP1_male_swarm_vel = sort_population(NP1_male_swarm_pos, NP1_male_swarm_vel)
    # Sorting for NP1 females
    NP1_female_fitness, NP1_female_swarm_pos, NP1_female_swarm_vel = sort_population(NP1_female_swarm_pos, NP1_female_sw
arm_vel)

    # Gbest updation
    gbest_fitness, gbest = update_gbest(gbest_fitness, gbest, NP1_male_fitness, NP1_male_swarm_pos, NP1_female_fitness,
NP1_female_swarm_pos)

    # Crossover and mutation for NP1
    NP1_male_swarm_pos, NP1_female_swarm_pos, NP1_male_swarm_vel, NP1_female_swarm_vel = crossover_and_mutation(NP1_male
_swarm_pos, NP1_female_swarm_pos, NP1_male_swarm_vel, NP1_female_swarm_vel, tot_features, subpop_size, l)

    # Updating swarm position for NP1
    update_swarm_position(NP1_male_swarm_pos, NP1_male_swarm_vel)
    update_swarm_position(NP1_female_swarm_pos, NP1_female_swarm_vel)

    # ELITISM: Save the elite individuals and Replace the worst individuals with the elite individuals for NP1
    elitism(NP1_male_swarm_pos, NP1_male_swarm_vel, NP1_female_swarm_pos, NP1_female_swarm_vel, elite_swarm_pos, elite_s
warm_fitness, num_elite, tot_features)

    # Updating gravity and nuptial dance for NP1
    d = d * delta
    fl = fl * delta

    # Gbest updation
    gbest_fitness, gbest = update_gbest(gbest_fitness, gbest, NP1_male_fitness, NP1_male_swarm_pos, NP1_female_fitness,
NP1_female_swarm_pos)

    ####################################################### NP2 ##########################################################

    # NP2 Males
    iterations_without_improvement = 0
    while iterations_without_improvement < max_neighbors:
        for i in range(subpop_size):
```

```
            if NP2_male_fitness[i] < gbest_fitness:
                gbest_fitness = NP2_male_fitness[i]
                gbest = NP2_male_swarm_pos[i].copy()

            if NP2_male_fitness[i] < pbest_fitness[i]:
                pbest_fitness[i] = NP2_male_fitness[i]
                pbest[i] = NP2_male_swarm_pos[i].copy()

            # Velocity updation for NP2 males
            if i == 0:
                for j in range(tot_features):
                    NP2_male_swarm_vel[0][j] = NP2_male_swarm_vel[0][j] + d * np.random.uniform(-1, 1)
            else:
                sum = 0
                for j in range(tot_features):
                    sum = sum + (NP2_male_swarm_pos[i][j] - gbest[j]) * (NP2_male_swarm_pos[i][j] - gbest[j])
                rg = math.sqrt(sum)
                sum = 0
                for j in range(tot_features):
                    sum = sum + (NP2_male_swarm_pos[i][j] - pbest[i][j]) * (NP2_male_swarm_pos[i][j] - pbest[i][j])
                rp = math.sqrt(sum)
                for j in range(tot_features):
                    NP2_male_swarm_vel[i][j] = g * NP2_male_swarm_vel[i][j] + a1 * math.exp(-beta * rp * rp) * (
                            pbest[i][j] - NP2_male_swarm_pos[i][j]) + a2 * math.exp(-beta * rg * rg) * (
                                                   gbest[j] - NP2_male_swarm_pos[i][j])
                    # Include attraction towards personal best (pbest)
                    NP2_male_swarm_vel[i][j] += a1 * (pbest[i][j] - NP2_male_swarm_pos[i][j])

                    # Include attraction towards global best (gbest)
                    NP2_male_swarm_vel[i][j] += a2 * (gbest[j] - NP2_male_swarm_pos[i][j])

            # Velocity updation for NP2 females
            if NP2_female_fitness[i] <= NP2_male_fitness[i]:
                sum = 0
                for j in range(tot_features):
                    sum = sum + (NP2_male_swarm_pos[i][j] - NP2_female_swarm_pos[i][j]) * (NP2_male_swarm_pos[i][j] - NP
2_female_swarm_pos[i][j])
                rmf = math.sqrt(sum)
                NP2_female_swarm_vel[i][j] = g * NP2_female_swarm_vel[i][j] + a2 * math.exp(-beta * rmf * rmf) * (NP2_ma
le_swarm_pos[i][j] - NP2_female_swarm_pos[i][j])
            else:
                for j in range(tot_features):
                    NP2_female_swarm_vel[i][j] = g * NP2_female_swarm_vel[i][j] + fl * np.random.uniform(-1, 1)

        # Levy flight for NP2 females
        for i in range(subpop_size):
            if NP2_female_fitness[i] > NP2_male_fitness[i]:
                # Employ Levy flight for exploration
                for j in range(tot_features):
                    NP2_female_swarm_pos[i][j] += levy_flight(lf_size)
            else:
                # Perform...
                for j in range(tot_features):
                    NP2_female_swarm_pos[i][j] = np.random.uniform(-1, 1)

        # Call migration operator for NP2 males
        NP2_male_swarm_pos = migration_operator(NP2_male_swarm_pos, NP2_female_swarm_pos, NP1_male_swarm_pos, 1.2, float
(5/12))
        # Call adjusting operator for NP2 females
        NP2_female_swarm_pos = adjusting_operator(NP2_female_swarm_pos, NP1_male_swarm_pos, itr)

        # Sorting and updating for NP2 males
        NP2_male_fitness, NP2_male_swarm_pos, NP2_male_swarm_vel = sort_population(NP2_male_swarm_pos, NP2_male_swarm_ve
l)
        # Sorting and updating for NP2 females
        NP2_female_fitness, NP2_female_swarm_pos, NP2_female_swarm_vel = sort_population(NP2_female_swarm_pos, NP2_femal
e_swarm_vel)

        #Gbest updation for NP2
        new_gbest_fitness, new_gbest = update_gbest(gbest_fitness, gbest, NP2_male_fitness, NP2_male_swarm_pos, NP2_fema
le_fitness, NP2_female_swarm_pos)
```

```
        ############# v2 ################################
        if new_gbest_fitness < gbest_fitness:
            gbest_fitness = new_gbest_fitness
            gbest = new_gbest.copy()
            print("Iteration:", itr, "GBest Fitness:", gbest_fitness)
            iterations_without_improvement = 0  # Reset the counter when a better solution is found
        else:
            print("Iteration:", itr, "New GBest Fitness:", new_gbest_fitness)
            iterations_without_improvement += 1  # Increment the counter when no improvement is found

        # Crossover and mutation for NP2
        NP2_male_swarm_pos, NP2_female_swarm_pos, NP2_male_swarm_vel, NP2_female_swarm_vel = crossover_and_mutation(NP2_
male_swarm_pos, NP2_female_swarm_pos, NP2_male_swarm_vel, NP2_female_swarm_vel, tot_features, subpop_size, l)

        # Updating swarm position for NP2
        update_swarm_position(NP2_male_swarm_pos, NP2_male_swarm_vel)
        update_swarm_position(NP2_female_swarm_pos, NP2_female_swarm_vel)

        # ELITISM: Save the elite individuals and Replace the worst individuals with the elite individuals for NP2
        elitism(NP2_male_swarm_pos, NP2_male_swarm_vel, NP2_female_swarm_pos, NP2_female_swarm_vel, elite_swarm_pos, eli
te_swarm_fitness, num_elite, tot_features)

        # Updating gravity and nuptial dance for NP2
        g = gmax-((gmax-gmin)*itr/max_iterations)
        d = d * delta
        fl = fl * delta

        #Gbest updation for NP2
        gbest_fitness, gbest = update_gbest(gbest_fitness, gbest, NP2_male_fitness, NP2_male_swarm_pos, NP2_female_fitne
ss, NP2_female_swarm_pos)

    # Append the current gbest_fitness to the convergence_curve list
    # convergence_curve.append(gbest_fitness)
    print("Iteration:", itr, "GBest Fitness:", gbest_fitness)

# Gbest = transfer_func(gbest)
selected_features = transfer_func(gbest)
for j in range(tot_features):
    if selected_features[j] >= 0.25:
        selected_features[j] = 1
    else:
        selected_features[j] = 0
number_of_selected_features = np.sum(selected_features)
print("NUM:", number_of_selected_features)
print("SELECTED FEATURES:", selected_features)

features = [df.columns[j] for j in range(len(selected_features)) if selected_features[j] == 1]
if not features:
    acc = 0
else:
    new_x_train = x_train[features]
    new_x_test = x_test[features]
    _classifier = SVC(random_state=seed)
    _classifier.fit(new_x_train, y_train)
    predictions = _classifier.predict(new_x_test)
    acc = accuracy_score(y_true=y_test, y_pred=predictions)
    fitness = acc
print("ACC:", acc)

end_time=process_time()
print("time in seconds =",(end_time-start_time))
print("time in microseconds =",(end_time-start_time) * 1e6)
```

APPENDIX C

MMO-KNN EXPERIMENTAL RESULTS FOR ALL 18 UCI DATASETS

Algorithm	Dataset	Pre FS Accuracy	Fitness Score	Selected Features Count	Selected Features	Pre FS Accuracy	Process time (s)	Process time (ms)
MMO-KNN	Breastcancer	0.6071428571428571	0.019142857142857093	5	[0. 1. 0. 1. 0. 1. 1. 0. 1. 0.]	0.9857142857142858	164.28125	164281250.0
		0.5571428571428572	0.02621428571428575	5	[0. 1. 1. 1. 0. 1. 0. 1. 0. 0.]	0.9785714285714285	167.203125	167203125.0
		0.6428571428571429	0.004	4.0	[0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	172.015625	172015625.0
		0.6571428571428571	0.027214285714285746	6	[0. 1. 0. 1. 1. 0. 1. 1. 0. 1.]	0.9785714285714285	176.140625	176140625.0
		0.5928571428571429	0.03835714285714285	3	[0. 1. 0. 0. 1. 0. 0. 0. 0. 1.]	0.9642857142857143	181.9375	181937500.0
		0.5642857142857143	0.011071428571428546	4	[0. 1. 1. 1. 0. 0. 0. 1. 0. 0.]	0.9928571428571429	177.34375	177343750.0
		0.7	0.018142857142857093	4	[0. 1. 1. 0. 0. 0. 1. 0. 0. 1.]	0.9857142857142858	170.140625	170140625.0
		0.5714285714285714	0.019142857142857093	5.0	[0. 1. 1. 0. 0. 1. 1. 1. 0. 0.]	0.9857142857142858	172.671875	172671875.0
		0.55	0.009071428571428546	2	[0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]	0.9928571428571429	164.484375	164484375.0
		0.6428571428571429	0.011071428571428546	4	[0. 1. 1. 0. 0. 0. 1. 0. 0. 1.]	0.9928571428571429	173.59375	173593750.0
		0.60857142857143		4.2		**0.98571428571429**	171.98125	
	BreastEW	0.9210526315789473	0.052771929824561456	2.0	[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]	0.9473684210526315	257.703125	257703125.0
		0.9122807017543859	0.035403508771929826	2.0	[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]	0.9649122807017544	238.828125	238828125.0
		0.8596491228070176	0.03640350877192982	5.0	[0. 0. 0. 0. 1. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]	0.9649122807017544	262.796875	262796875.0
		0.9122807017543859	0.019701754385964967	7.0	[0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]	0.9824561403508771	268.28125	268281250.0
		0.8859649122807017	0.06212280701754383	4.0	[0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]	0.9385964912280702	258.28125	258281250.0
		0.92982456140350	0.026719298245	2.0	[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.	0.9736842105263158	235.8125	235812500.0

		88	614004		0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]			
		0.9122807017543859	0.01001754385964915	4.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0.]	0.9912280701754386	249.5625	249562500.0
		0.9035087719298246	0.05310526315789479	3.0	[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9473684210526315	267.0	267000000.0
		0.8947368421052632	0.04442105263157897	3.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0.]	0.956140350877193	233.390625	233390625.0
		0.9385964912280702	0.018368421052631635	3.0	[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0.]	0.9824561403508771	264.46875	264468750.0
		0.90701754385965		3.5		**0.96491228070175**	253.6125	
	CongressEW	0.896551724137931	0.039137931034482716	8.0	[0. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 0.]	0.9655172413793104	149.515625	149515625.0
		0.9425287356321839	0.024633620689655147	3.0	[0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]	0.9770114942528736	145.0625	145062500.0
		0.9310344827586207	0.013879310344827573	4.0	[0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]	0.9885057471264368	146.46875	146468750.0
		0.9425287356321839	0.013879310344827573	4.0	[0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1.]	0.9885057471264368	152.796875	152796875.0
		0.9080459770114943	0.026508620689655146	6.0	[0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.]	0.9770114942528736	160.609375	160609375.0
		0.9195402298850575	0.0025	4.0	[0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0.]	1.0	143.765625	143765625.0
		0.9310344827586207	0.013879310344827573	4.0	[0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.]	0.9885057471264368	147.171875	147171875.0
		0.9770114942528736	0.015129310344827573	6.0	[1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0.]	0.9885057471264368	148.25	148250000.0
		0.8850574712643678	0.05939655172413797	4.0	[0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9425287356321839	143.609375	143609375.0
		0.9195402298850575	0.03788793103448272	6.0	[0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1.]	0.9655172413793104	154.421875	154421875.0
		0.92528735632184		4.9		**0.97816091954023**	149.1671875	
	Exactly	0.73	0.020234615384615396	7.0	[1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	0.985	236.5	236500000.0
		0.705	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	239.0625	239062500.0
		0.74	0.005384615384615384	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0.]	1.0	243.96875	243968750.0
		0.705	0.01033461538461539	7.0	[1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	0.995	238.0	238000000.0
		0.76	0.015284615384615393	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]	0.99	239.078125	239078125.0

		0.715	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	246.8125	246812500.0
		0.74	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	232.71875	232718750.0
		0.695	0.005384615384615384	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0.]	1.0	237.90625	237906250.0
		0.68	0.020234615384615396	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]	0.985	226.078125	226078125.0
		0.795	0.025184615384615403	7.0	[1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	0.98	224.71875	224718750.0
		0.7265		6.7		**0.9935**	236.484375	
	Exactly2	0.72	0.2190038461538461	8.0	[1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0.]	0.785	247.140625	247140625.0
		0.725	0.2091038461538461	8.0	[1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 0. 1. 0.]	0.795	243.359375	243359375.0
		0.735	0.20833461538461534	7.0	[1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0.]	0.795	274.234375	274234375.0
		0.73	0.20261538461538456	6.0	[1. 0. 1. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0.]	0.8	254.890625	254890625.0
		0.705	0.21141153846153843	11.0	[1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1.]	0.795	237.84375	237843750.0
		0.74	0.19117692307692302	4.0	[0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0.]	0.81	256.9375	256937500.0
		0.715	0.2018461538461538	5.0	[0. 1. 0. 0. 0. 1. 1. 0. 1. 0. 0. 1. 0.]	0.8	239.234375	239234375.0
		0.74	0.2117461538461538	5.0	[1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0.]	0.79	249.578125	249578125.0
		0.74	0.20833461538461534	7.0	[0. 1. 1. 1. 1. 0. 0. 1. 0. 0. 1. 1. 0.]	0.795	251.296875	251296875.0
		0.745	0.20492307692307687	9.0	[0. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0.]	0.8	255.53125	255531250.0
		0.7295		7		**0.7965**	251.0046875	
	HeartEW	0.7962962962962963	0.13064102564102567	3.0	[0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1.]	0.8703703703703703	120.6875	120687500.0
		0.6666666666666666	0.11307692307692312	4.0	[0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0.]	0.8888888888888888	125.0625	125062500.0
		0.6851851851851852	0.11384615384615389	5.0	[0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 1. 0.]	0.8888888888888888	117.734375	117734375.0
		0.6296296296296297	0.16961538461538456	6.0	[1. 1. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1.]	0.8333333333333334	125.28125	125281250.0
		0.7222222222222222	0.07717948717948718	5.0	[0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1.]	0.9259259259259259	127.75	127750000.0
		0.7037037037037037	0.16807692307692304	4.0	[0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1.]	0.8333333333333334	122.71875	122718750.0
		0.6666666666666666	0.15128205128205127	6.0	[0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0.]	0.8518518518518519	117.265625	117265625.0

		0.6296296296296297	0.11384615384615389	5.0	[0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1.]	0.8888888888888888	125.59375	125593750.0
		0.7962962962962963	0.07641025641025641	4.0	[0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0.]	0.9259259259259259	127.046875	127046875.0
		0.7222222222222222	0.07717948717948718	5.0	[0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1.]	0.9259259259259259	120.40625	120406250.0
		0.70185185185185		4.7		**0.8833333333333**	122.9546875	
	IonosphereEW	0.8428571428571429	0.0014705882352941176	5.0	[0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1.]	1.0	235.8125	235812500.0
		0.8428571428571429	0.05892436974789918	8.0	[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0.]	0.9428571428571428	249.65625	249656250.0
		0.8571428571428571	0.015319327731092387	4.0	[0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]	0.9857142857142858	204.015625	204015625.0
		0.8571428571428571	0.04331092436974786	3.0	[0. 0. 1. 0. 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.]	0.9571428571428572	206.09375	206093750.0
		0.8571428571428571	0.0005882352941176471	2.0	[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	1.0	210.0	210000000.0
		0.8142857142857143	0.057747899159663885	4.0	[1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9428571428571428	221.046875	221046875.0
		0.7857142857142857	0.043899159663865504	5.0	[0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0.]	0.9571428571428572	232.046875	232046875.0
		0.8428571428571429	0.015025210084033564	3.0	[0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9857142857142858	216.484375	216484375.0
		0.8142857142857143	0.029168067226890766	3.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9714285714285714	209.109375	209109375.0
		0.9	0.015319327731092387	4.0	[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9857142857142858	204.765625	204765625.0
		0.84142857142857		4.1		**0.97285714285714**	218.903125	
	KrVsKpEW	0.9624413145539906	0.02303599374021914	16.0	[1. 0. 0. 1. 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1.]	0.9812206572769953	1059.921875	1059921875.0
		0.9561815336463224	0.02204225352112679	18.0	[1. 0. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 1.]	0.9827856025039123	958.140625	958140625.0
		0.96400625978090	0.028079029733	23	[1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0.	0.9780907668231612	948.34375	948343750.0

		77	959295		1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1.]			
		0.9655712050078247	0.025418622848200272	19.0	[1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0.]	0.9796557120500783	985.515625	985515625.0
		0.9608763693270735	0.03566510172143971	28.0	[1. 0. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1.]	0.971830985915493	946.359375	946359375.0
		0.9655712050078247	0.02708528951486694	25.0	[1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0.]	0.9796557120500783	934.265625	934265625.0
		0.9671361502347418	0.024820031298904567	28.0	[1. 0. 0. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]	0.9827856025039123	983.390625	983390625.0
		0.9546165884194053	0.024820031298904567	28.0	[1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.]	0.9827856025039123	969.671875	969671875.0
		0.9561815336463224	0.01695618153364637	22.0	[1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 0.]	0.9890453834115805	921.40625	921406250.0
		0.9608763693270735	0.017511737089201923	24.0	[1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1.]	0.9890453834115805	1025.765625	1025765625.0
		0.96134585289515		23.1		**0.98169014084507**	973.278125	
	Lymphography	0.7333333333333333	0.005	9.0	[1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 0.]	1.0	111.265625	111265625.0
		0.6333333333333333	0.06822222222222221	4.0	[0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0.]	0.9333333333333333	111.890625	111890625.0
		0.8	0.06766666666666665	3.0	[0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0.]	0.9333333333333333	112.375	112375000.0
		0.7666666666666667	0.06877777777777777	5.0	[0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0.]	0.9333333333333333	112.203125	112203125.0
		0.7666666666666667	0.10344444444444442	8.0	[0. 1. 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1.]	0.9	111.8125	111812500.0
		0.7666666666666667	0.03688888888888888	7.0	[0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 0.]	0.9666666666666667	117.671875	117671875.0
		0.8666666666666667	0.0391111111111111	11.0	[0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 0. 0. 0. 1. 1.]	0.9666666666666667	124.828125	124828125.0
		0.7	0.07044444444444443	8.0	[0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0.]	0.9333333333333333	132.859375	132859375.0
		0.8	0.03577777777777777	5.0	[0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1.]	0.9666666666666667	122.28125	122281250.0
		0.8333333333333334	0.03855555555555555	10.0	[0. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 1. 0.]	0.9666666666666667	103.46875	103468750.0
		0.76666666666667		7		**0.95**	116.065625	

	M-of-N	0.88	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	216.171875	216171875.0
		0.905	0.011103846153846159	8.0	[1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0.]	0.995	208.8125	208812500.0
		0.86	0.021003846153846168	8.0	[1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 1.]	0.985	227.078125	227078125.0
		0.905	0.01033461538461539	7.0	[1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	0.995	243.25	243250000.0
		0.925	0.005384615384615384	7.0	[1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	238.984375	238984375.0
		0.895	0.005384615384615384	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0.]	1.0	169.078125	169078125.0
		0.905	0.005384615384615384	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0.]	1.0	220.28125	220281250.0
		0.85	0.005384615384615384	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0.]	1.0	224.484375	224484375.0
		0.875	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	260.453125	260453125.0
		0.895	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	213.09375	213093750.0
		0.8895		6.9		**0.9975**	222.16875	
	PenglungEW	0.7333333333333333	0.0008923076923076922	29.0	[0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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. 1. 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. 0. 0. 0. 1. 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. 1. 0. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.	1.0	195.65625	195656250.0

					0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]			
		0.8	0.0009230769230769232	30.0	[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. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 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. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 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. 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. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	1.0	185.9375	185937500.0
		0.7333333333333333	0.0015076923076923078	49.0	[0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 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. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.	1.0	194.078125	194078125.0

					0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 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.]			
		0.9333333333333333	0.0006769230769230769	22.0	[0. 0. 0. 0. 0. 1. 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. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 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. 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. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 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. 0. 0. 0. 1. 0. 0. 0. 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. 0. 0. 0. 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.]	1.0	174.328125	174328125.0
		1.0	0.0007692307692307693	25.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.	1.0	176.484375	176484375.0

					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. 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. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 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. 1. 0. 0. 1. 0. 0. 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. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]			
		0.6666666666666666	0.0670153846153846	33.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 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. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]	0.9333333333333333	202.421875	202421875.0
		0.8	0.06679999999999998	26.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.	0.9333333333333333	185.375	185375000.0

					0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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. 1. 0. 1. 0. 0. 0. 0. 0. 0. 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. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 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. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]			
		0.8	0.067076923076 92306	35.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.	0.9333333333333333	196.0625	196062500.0

					0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]			
		0.8	0.001076923076923077	35.0	[0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 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. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]	1.0	192.796875	192796875.0
		0.6666666666666666	0.0671076923076923	36.0	[0. 1. 0. 0. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 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. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0.	0.9333333333333333	208.765625	208765625.0

					0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 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. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]			
		0.79333333333333		32		0.97333333333333	191.190625	
	Sonar	0.83333333333333 34	0.025571428571 4286	12.0	[0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0.]	0.9761904761904762	192.890625	192890625.0
		0.73809523809523 81	0.049142857142 857196	12.0	[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0.]	0.9523809523809523	193.15625	193156250.0
		0.76190476190476 19	0.025071428571 4286	9.0	[0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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. 1. 0. 1. 0. 0. 1.]	0.9761904761904762	174.65625	174656250.0
		0.83333333333333 34	0.026571428571 428597	18.0	[1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0.]	0.9761904761904762	197.53125	197531250.0
		0.83333333333333 34	0.025238095238 095264	10.0	[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]	0.9761904761904762	205.28125	205281250.0
		0.83333333333333 34	0.025071428571 4286	9.0	[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0.]	0.9761904761904762	185.671875	185671875.0
		0.73809523809523 81	0.024738095238 095264	7	[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]	0.9761904761904762	172.171875	172171875.0

		0.8809523809523809	0.005833333333333334	35.0	[0. 1. 0. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 0. 0.]	1.0	188.0	188000000.0
		0.8095238095238095	0.0250714285714286	9.0	[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9761904761904762	201.09375	201093750.0
		0.8571428571428571	0.025238095238095264	10.0	[1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0.]	0.9761904761904762	205.046875	205046875.0
		0.81190476190476		13.1		0.97619047619048	191.55	
	SpectEW	0.8518518518518519	0.09530303030303026	8.0	[1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 0. 0.]	0.9074074074074074	146.28125	146281250.0
		0.7777777777777778	0.06000000000000002	11.0	[1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1.]	0.9444444444444444	153.53125	153531250.0
		0.8148148148148148	0.0953030303030302	8.0	[1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1.]	0.9074074074074074	155.75	155750000.0
		0.7592592592592593	0.11454545454545459	10.0	[1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 1. 1. 0.]	0.8888888888888888	152.890625	152890625.0
		0.7962962962962963	0.11272727272727277	6.0	[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]	0.8888888888888888	152.53125	152531250.0
		0.7592592592592593	0.1503030303030303	8.0	[0. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1.]	0.8518518518518519	151.359375	151359375.0 .
		0.9259259259259259	0.024242424242424215	13.0	[0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 1. 0.]	0.9814814814814815	150.984375	150984375.0
		0.7407407407407407	0.11318181818181823	7.0	[0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0.]	0.8888888888888888	129.0	129000000.0
		0.7777777777777778	0.09348484848484845	4.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.]	0.9074074074074074	117.484375	117484375.0
		0.7777777777777778	0.13106060606060607	6.0	[0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1.]	0.8703703703703703	9.109375	9109375.0
		0.79814814814815		8.1		0.9037037037037	131.8921875	
	Tic-tac-toe	0.8541666666666666	0.15437500000000004	9	[1. 1. 1. 1. 1. 1. 1. 1. 1.]	0.8541666666666666	233.671875	233671875.0
		0.859375	0.14921875	9	[1. 1. 1. 1. 1. 1. 1. 1. 1.]	0.859375	221.328125	221328125.0
		0.8229166666666666	0.18531250000000005	9	[1. 1. 1. 1. 1. 1. 1. 1. 1.]	0.8229166666666666	237.015625	237015625.0
		0.8072916666666666	0.1985590277777778	7	[1. 0. 1. 1. 1. 1. 1. 0. 1.]	0.8072916666666666	216.703125	216703125.0
		0.83333333333333	0.165399305555	5	[1. 0. 0. 1. 1. 0. 1. 0. 1.]	0.8385416666666666	239.640625	239640625.0

		34	5556					
		0.8385416666666666	0.16984375000000004	9	[1. 1. 1. 1. 1. 1. 1. 1. 1.]	0.8385416666666666	227.515625	227515625.0
		0.828125	0.1653993055555556	5	[1, 0, 1, 1, 1, 0, 1, 0, 0]	0.8385416666666666	219.484375	219484375.0
		0.8541666666666666	0.15437500000000004	9	[1, 1, 1, 1, 1, 1, 1, 1, 1]	0.8541666666666666	218.703125	218703125.0
		0.8072916666666666	0.18602430555555552	5	[0, 0, 0, 1, 1, 0, 1, 1, 1]	0.8177083333333334	223.046875	223046875.0
		0.8489583333333334	0.15953124999999996	9	[1, 1, 1, 1, 1, 1, 1, 1, 1]	0.8489583333333334	226.765625	226765625.0
		0.83541666666667		7.6		0.83802083333333	226.3875	
	Vote	0.9333333333333333	0.01837500000000005	3.0	[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0.]	0.9833333333333333	130.3125	130312500.0
		0.8833333333333333	0.01837500000000005	3.0	[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0.]	0.9833333333333333	122.203125	122203125.0
		0.9666666666666667	0.00375	6.0	[1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1.]	1.0	125.359375	125359375.0
		0.8833333333333333	0.019625000000000052	5.0	[0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0.]	0.9833333333333333	120.875	120875000.0
		0.9166666666666666	0.001875	3.0	[0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0.]	1.0	119.71875	119718750.0
		0.8666666666666667	0.051375000000000046	3.0	[0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]	0.95	119.8125	119812500.0
		0.95	0.001875	3.0	[0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]	1.0	125.734375	125734375.0
		0.8833333333333333	0.019625000000000052	5.0	[0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0.]	0.9833333333333333	117.265625	117265625.0
		0.9666666666666667	0.001875	3.0	[0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]	1.0	129.234375	129234375.0
		0.9166666666666666	0.00125	2.0	[1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	1.0	119.9375	119937500.0
		0.91666666666667		3.6		0.98833333333333	123.0453125	
	WaveformEW	0.807	0.17775000000000005	18.0	[1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]	0.825	2875.1875	2875187500.0
		0.824	0.15253000000000003	28.0	[1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 0. 0.]	0.853	2814.46875	2814468750.0
		0.812	0.16466000000000003	29.0	[1. 1. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1.]	0.841	2766.609375	2766609375.0
		0.818	0.167880000000	30.0	[1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.	0.838	3397.609375	3397609375.0

			00003		1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1.]			.0
		0.804	0.17134000000000005	28.0	[0. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 1. 0.]	0.834	3137.71875	3137718750.0
		0.805	0.16812000000000005	27.0	[1. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0.]	0.837	3144.15625	3144156250.0
		0.813	0.16813000000000003	31.0	[0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 1.]	0.838	2628.046875	2628046875.0
		0.791	0.16985000000000003	26.0	[1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1.]	0.835	2943.96875	2943968750.0
		0.795	0.17035000000000003	28.0	[0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0.]	0.835	3252.125	3252125000.0
		0.794	0.17827000000000004	28.0	[1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0.]	0.827	2979.875	2979875000.0
		0.8063		27.3		0.8363	2993.9765625	
	Wine	0.7222222222222222	0.002307692307692308	3.0	[1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0.]	1.0	104.359375	104359375.0
		0.6944444444444444	0.002307692307692308	3.0	[0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]	1.0	105.515625	105515625.0
		0.75	0.0038461538461538464	5.0	[1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0.]	1.0	101.296875	101296875.0
		0.6944444444444444	0.03057692307692309	4.0	[1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0.]	0.9722222222222222	104.21875	104218750.0
		0.7777777777777778	0.029038461538461548	2.0	[1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]	0.9722222222222222	100.859375	100859375.0
		0.6944444444444444	0.02980769230769232	3.0	[1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0.]	0.9722222222222222	102.625	102625000.0
		0.6388888888888888	0.002307692307692308	3.0	[1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0.]	1.0	101.578125	101578125.0
		0.75	0.03057692307692309	4.0	[1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0.]	0.9722222222222222	104.75	104750000.0
		0.7777777777777778	0.0030769230769230774	4.0	[1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0.]	1.0	103.0	103000000.0

		0.6388888888888888	0.03211538461538463	6.0	[0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 0.]	0.9722222222222222	101.9375	101937500.0
		0.71388888888889		3.7		0.98611111111111	103.0140625	
	Zoo	0.85	0.052000000000000046	4.0	[0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.]	0.95	97.09375	97093750.0
		0.8	0.05325000000000005	6.0	[0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1.]	0.95	94.734375	94734375.0
		0.85	0.0025	4.0	[0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]	1.0	90.28125	90281250.0
		0.95	0.00375	6.0	[0. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 1. 0. 1. 0. 0.]	1.0	87.890625	87890625.0
		0.75	0.00375	6	[0. 0. 0. 1. 1. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0.]	1.0	88.828125	88828125.0
		0.85	0.0025	4.0	[0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0.]	1.0	117.046875	117046875.0
		0.9	0.0025	4.0	[1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0.]	1.0	113.5	113500000.0
		0.95	0.051375000000000046	3.0	[1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]	0.95	116.453125	116453125.0
		0.9	0.051375000000000046	3.0	[1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]	0.95	93.171875	93171875.0
		0.9	0.052000000000000046	4.0	[0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]	0.95	93.78125	93781250.0
		0.87		4.4		0.97777777777778	99.278125	

APPENDIX D

MMO-SVM EXPERIMENTAL RESULTS FOR ALL 18 UCI DATASETS

Algorithm	Dataset	Pre FS Accuracy	Fitness Score	Selected Features Count	Selected Feautes	Post FS Accuracy	Process time (s)	Process time (ms)
MMO-SVM	Breastcancer	0.6571428571428571	0.021142857142857092	7.0	[0. 1. 0. 1. 1. 0. 1. 1. 1. 1.]	0.9857142857142858	182.453125	152453125.0
		0.6571428571428571	0.037357142857142846	2.0	[0. 1. 1. 0. 0. 0. 0. 0. 0. 0.]	0.9642857142857143	197.796875	157796875.0
		0.6571428571428571	0.010071428571428547	3.0	[0. 1. 1. 0. 0. 0. 0. 1. 0. 0.]	0.9928571428571429	186.296875	156296875.0
		0.6571428571428571	0.025214285714285748	4.0	[0. 1. 1. 0. 1. 0. 1. 0. 0. 0.]	0.9785714285714285	189.796875	159796875.0
		0.6571428571428571	0.04542857142857139	3.0	[0. 1. 0. 0. 1. 0. 0. 0. 0. 1.]	0.9571428571428572	177.046875	137046875.0
		0.6571428571428571	0.010071428571428547	3.0	[0. 0. 1. 1. 0. 0. 0. 1. 0. 0.]	0.9928571428571429	175.328125	145328125.0
		0.6571428571428571	0.024214285714285747	3.0	[0. 0. 0. 1. 0. 1. 0. 0. 1. 0.]	0.9785714285714285	180.421875	150421875.0
		0.6571428571428571	0.025214285714285748	4.0	[0. 0. 1. 0. 1. 1. 0. 0. 1. 0.]	0.9785714285714285	166.0	166000000.0
		0.6571428571428571	0.01614285714285709	2.0	[0. 1. 1. 0. 0. 0. 0. 0. 0. 0.]	0.9857142857142858	199.921875	149921875.0
		0.6571428571428571	0.013071428571428546	6.0	[0. 1. 1. 0. 0. 1. 1. 1. 0. 1.]	0.9928571428571429	178.46875	158468750.0
		0.65714285714286		3.7		0.98071428571429	183.353125	
	BreastEW	0.9385964912280702	0.04442105263157897	3.0	[0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]	0.956140350877193	218.171875	218171875.0
		0.9298245614035088	0.02738596491228067	4.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]	0.9736842105263158	179.171875	179171875.0
		0.8508771929824561	0.04442105263157897	3.0	[0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]	0.956140350877193	176.359375	176359375.0
		0.9122807017543859	0.03640350877192982	5.0	[0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1.]	0.9649122807017544	207.109375	207109375.0
		0.8771929824561403	0.07047368421052631	3.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9298245614035088	179.3125	179312500.0
		0.92982456140350	0.035403508771	2.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.	0.9649122807017544	187.328125	187328125.0

		88	929826		0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]			
		0.9210526315789473	0.026719298245614004	2.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0.]	0.9736842105263158	180.390625	180390625.0
		0.8947368421052632	0.05510526315789479	9.0	[0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1.]	0.9473684210526315	175.03125	175031250.0
		0.9210526315789473	0.03607017543859649	4.0	[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9649122807017544	184.078125	184078125.0
		0.9298245614035088	0.02705263157894734	3.0	[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9736842105263158	177.453125	177453125.0
		0.91052631578947		3.8		0.96052631578947	186.440625	
	CongressEW	0.9310344827586207	0.03726293103448272	5.0	[0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0.]	0.9655172413793104	133.53125	133531250.0
		0.9655172413793104	0.024633620689655147	3.0	[0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]	0.9770114942528736	129.75	129750000.0
		0.9655172413793104	0.013879310344827573	4.0	[0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0.]	0.9885057471264368	133.125	133125000.0
		0.9770114942528736	0.024633620689655147	3.0	[0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9770114942528736	143.1875	143187500.0
		0.9425287356321839	0.0486422413793104	5.0	[0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0.]	0.9540229885057471	134.90625	134906250.0
		0.9770114942528736	0.001875	3.0	[0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]	1.0	129.828125	129828125.0
		0.9770114942528736	0.006875	11.0	[0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 0. 0.]	1.0	128.109375	128109375.0
		0.9770114942528736	0.01700431034482757	9.0	[1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0.]	0.9885057471264368	138.078125	138078125.0
		0.9080459770114943	0.05939655172413797	4.0	[0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0.]	0.9425287356321839	131.4375	131437500.0
		0.9425287356321839	0.03663793103448272	4.0	[0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.]	0.9655172413793104	132.9375	132937500.0
		0.95632183908046		5.1		0.97586206896552	133.4890625	
	Exactly	0.735	0.024415384615384634	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	0.98	449.8125	449812500.0
		0.7	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	449.375	449375000.0
		0.71	0.044215384615384656	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	0.96	461.21875	461218750.0
		0.7	0.009565384615384621	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	0.995	461.203125	461203125.0

		0.735	0.03431538461538464	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	0.97	471.90625	471906250.0
		0.725	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	426.40625	426406250.0.0
		0.73	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	433.921875	433921875.0
		0.695	0.04916538461538466	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	0.955	454.0625	454062500.0
		0.67	0.06478461538461544	7.0	[1. 0. 1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 0.]	0.94	417.484375	417484375.0
		0.72	0.07963461538461535	7.0	[1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	0.925	477.453125	477453125.0
		0.712		6.2		0.9725	450.284375	
	Exactly2	0.765	0.2323153846153846	6.0	[0. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 0.]	0.77	356.796875	356796875.0
		0.76	0.2323153846153846	8.0	[0. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1.]	0.77	342.3125	342312500.0
		0.765	0.23121153846153844	11.0	[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0.]	0.775	351.1875	351187500.0
		0.755	0.2323153846153846	6.0	[1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 1.]	0.77	367.90625	367906250.0
		0.765	0.22549230769230766	10.0	[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0.]	0.78	636.875	636875000.0
		0.76	0.22736538461538458	6.0	[0. 0. 0. 1. 0. 1. 1. 1. 1. 1. 0. 0. 0.]	0.775	369.640625	369640625.0
		0.765	0.2323153846153846	6.0	[0. 0. 0. 1. 0. 1. 1. 1. 1. 1. 0. 0. 0.]	0.77	348.84375	348843750.0
		0.75	0.22736538461538458	6.0	[0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0.]	0.775	339.78125	339781250.0
		0.76	0.22813461538461535	7.0	[1. 0. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0.]	0.775	360.078125	360078125.0
		0.765	0.2323153846153846	6.0	[0. 0. 0. 1. 0. 1. 1. 1. 1. 1. 0. 0. 0.]	0.77	339.28125	339281250.0
		0.761		7.2		0.773	381.2703125	
	HeartEW	0.7407407407407407	0.13141025641025644	4.0	[0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.]	0.8703703703703703	99.890625	99890625.0
		0.6296296296296297	0.11384615384615389	5.0	[0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0.]	0.8888888888888888	108.703125	108703125.0
		0.6851851851851852	0.11230769230769234	3.0	[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0.]	0.8888888888888888	101.078125	101078125.0
		0.6481481481481481	0.16807692307692304	4.0	[0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0.]	0.8333333333333334	103.625	103625000.0
		0.6666666666666666	0.07717948717948718	5.0	[0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1.]	0.9259259259259259	100.484375	100484375.0
		0.7222222222222222	0.132948717948	6.0	[0. 1. 1. 0. 0. 1. 1. 0. 1. 1. 0. 0. 0.]	0.8703703703703703	102.78125	102781250.0

		22	71796					
		0.5740740740740741	0.11307692307692312	4.0	[1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1.]	0.8888888888888888	103.1875	103187500.0
		0.7037037037037037	0.11538461538461543	7.0	[0. 0. 0. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1.]	0.8888888888888888	107.09375	107093750.0
		0.6481481481481481	0.05961538461538464	6.0	[0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0.]	0.9444444444444444	101.515625	101515625.0
		0.5555555555555556	0.07641025641025641	4.0	[0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0.]	0.9259259259259259	107.65625	107656250.0
		0.65740740740741		4.8		0.89259259259259	103.6015625	
	Ionosphere	0.9857142857142858	0.0017647058823529412	6.0	[1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	1.0	111.015625	111015625.0
		0.9285714285714286	0.030638655462184885	8.0	[0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]	0.9714285714285714	149.484375	149484375.0
		0.9428571428571428	0.015907563025210035	6.0	[0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9857142857142858	134.8125	134812500.0
		0.9142857142857143	0.016201680672268858	7.0	[0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]	0.9857142857142858	230.828125	230828125.0
		0.9428571428571428	0.0023529411764705885	8.0	[0. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0.]	1.0	130.921875	130921875.0
		0.9142857142857143	0.043899159663865504	5.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1.]	0.9571428571428572	138.734375	138734375.0
		0.9571428571428572	0.0026470588235294116	9.0	[1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0.]	1.0	153.328125	153328125.0
		0.9571428571428572	0.0017647058823529412	6.0	[0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]	1.0	141.890625	141890625.0
		0.9285714285714286	0.015319327731092387	4.0	[0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9857142857142858	163.828125	163828125.0
		0.9714285714285714	0.002058823529411765	7.0	[1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.]	1.0	163.75	163750000.0
		0.94428571428571		6.6		0.98857142857143	151.859375	
	KrVsKpEW	0.9765258215962441	0.011870109546165842	26.0	[1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 0.]	0.9953051643192489	3670.09375	3670093750.0

		0.974960876369327	0.017511737089201923	24.0	[1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0.]	0.9890453834115805	3718.296875	3718296875.0
		0.974960876369327	0.01961658841940528	26.0	[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1.]	0.9874804381846636	3529.5	3529500000.0
		0.9640062597809077	0.0330046948356808	24.0	[1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 1. 1. 1.]	0.97339593114241	3961.84375	3961843750.0
		0.97339593114241	0.018783255086071947	23.0	[1. 1. 0. 1. 0. 1. 1. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0.]	0.9874804381846636	3632.625	3632625000.0
		0.9843505477308294	0.014690923317683883	25.0	[1. 0. 0. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0.]	0.9953051643192489	3368.75	3368750000.0
		0.9812206572769953	0.017233959311424147	23.0	[1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 1. 0. 1. 0.]	0.9890453834115805	3993.9375	3993937500.0
		0.9780907668231612	0.018067292644757482	26.0	[1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1.]	0.9890453834115805	3633.015625	3633015625.0
		0.9702660406885759	0.025813771517996915	26.0	[1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0.]	0.9812206572769953	3268.6875	3268687500.0
		0.9796557120500783	0.013697183098591529	27.0	[1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0.]	0.9937402190923318	3744.578125	3744578125.0
		0.97574334898279		25		0.98810641627543	3652.1328125	
	Lymphography	0.8333333333333334	0.03466666666666666	3.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1.]	0.9666666666666667	112.5	112500000.0
		0.6	0.06822222222222221	4.0	[0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]	0.9333333333333333	121.765625	121765625.0
		0.8	0.06766666666666665	3.0	[0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9333333333333333	110.109375	110109375.0
		0.8	0.1012222222222222	4.0	[0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0.]	0.9	111.375	111375000.0
		0.7666666666666667	0.1331111111111111	2.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1.]	0.8666666666666667	104.359375	104359375.0
		0.8	0.07155555555555554	10.0	[0. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 1. 1. 0.]	0.9333333333333333	107.546875	107546875.0
		0.8666666666666667	0.10011111111111108	2.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0.]	0.9	107.46875	107468750.0
		0.76666666666666	0.068222222222	4.0	[0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0.	0.9333333333333333	109.421875	109421875.0

		67	22221		0. 1. 0. 0. 0.]			
		0.8333333333333334	0.06933333333333332	6.0	[0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 1.]	0.9333333333333333	111.328125	111328125.0
		0.7666666666666667	0.06822222222222221	4.0	[0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0.]	0.9333333333333333	106.859375	106859375.0
		0.78333333333333		4.2		0.92333333333333	110.2734375	
	M-of-n	1.0	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	404.0	404000000.0
		1.0	0.005384615384615384	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0.]	1.0	469.921875	469921875.0
		1.0	0.005384615384615384	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0.]	1.0	454.8125	454812500.0
		1.0	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	481.921875	481921875.0
		1.0	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	460.046875	460046875.0
		1.0	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	491.546875	491546875.0
		1.0	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	412.71875	412718750.0
		1.0	0.004615384615384616	6.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]	1.0	419.875	419875000.0
		1.0	0.005384615384615384	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0.]	1.0	403.5	403500000.0
		1.0	0.005384615384615384	7.0	[1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0.]	1.0	420.15625	420156250.0
		1		6.4		1.0	441.85	
	PenglungEW	0.8	0.06723076923076922	40.0	[0. 0. 0. 0. 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. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.	0.9333333333333333	138.40625	138406250.0

					1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0.]			
		0.73333333333333 33	0.000861538461 5384615	28.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 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. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 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. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]	1.0	137.828125	137828125.0
		0.8	0.000646153846 1538462	21.0	[0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 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.	1.0	138.53125	138531250.0

					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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 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. 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.]			
		0.86666666666666 67	0.000830769230 7692307	27.0	[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 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. 0. 1. 0. 0. 0. 0. 1. 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. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	1.0	160.640625	160640625.0
		0.8	0.001476923076 923077	48.0	[1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 0. 0. 0. 1. 1. 1.	1.0	160.8125	160812500.0

					0. 1. 1. 0. 1. 1. 0. 0. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0.]			
		0.6	0.066707692307 6923	23.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 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. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 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. 0. 0. 0. 0. 0.	0.9333333333333333	161.015625	161015625.0

					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.]			
		0.8	0.06655384615384614	18.0	[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 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. 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. 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. 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. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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.]	0.9333333333333333	153.171875	153171875.0
		0.6666666666666666	0.06639999999999999	13.0	[0. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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. 1. 0. 0. 0. 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. 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. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.	0.9333333333333333	161.328125	161328125.0

					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. 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. 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.]			
		0.8	0.000738461538 4615385	24.0	[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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 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. 1. 1. 0. 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. 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. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 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. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	1.0	131.109375	131109375.0
		0.66666666666666 66	0.132461538461 53844	15.0	[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. 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. 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. 1. 0. 0. 0. 0. 0. 0.	0.8666666666666667	137.625	137625000.0

					0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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. 1. 0. 1. 0. 0. 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. 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. 0. 0. 1. 0. 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. 0. 0. 0. 0. 1. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]			
		0.753333333333333		25.7		0.96	148.046875	
	Sonar	0.7857142857142857	0.07254761904761903	11.0	[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9285714285714286	122.0	122000000.0
		0.8095238095238095	0.07188095238095235	7.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9285714285714286	131.71875	131718750.0
		0.8571428571428571	0.003333333333333333	20.0	[1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]	1.0	125.1875	125187500.0
		0.7857142857142857	0.04897619047619053	11.0	[1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9523809523809523	126.765625	126765625.0
		0.9047619047619048	0.026571428571428597	18.0	[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1.]	0.9761904761904762	107.984375	107984375.0
		0.8333333333333334	0.04980952380952386	16.0	[1. 0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0.	0.9523809523809523	108.1875	108187500.0

					0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1.]			
		0.8095238095238095	0.04880952380952386	10.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]	0.9523809523809523	105.15625	105156250.0
		0.8809523809523809	0.04930952380952386	13.0	[0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]	0.9523809523809523	116.28125	116281250.0
		0.8333333333333334	0.02490476190476193	8.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0.]	0.9761904761904762	104.8125	104812500.0
		0.7857142857142857	0.048309523809523865	7.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	0.9523809523809523	116.15625	116156250.0
		0.82857142857143		12.1		0.95714285714286	116.425	
	SpectEW	0.8703703703703703	0.11227272727272732	5.0	[1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.]	0.8888888888888888	90.84375	90843750.0
		0.8703703703703703	0.07560606060606061	5.0	[0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]	0.9259259259259259	82.21875	82218750.0
		0.8148148148148148	0.11181818181818186	4.0	[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]	0.8888888888888888	82.234375	82234375.0
		0.8148148148148148	0.14893939393939393	5.0	[0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]	0.8518518518518519	88.3125	88312500.0
		0.8148148148148148	0.11272727272727277	6.0	[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 0. 1.]	0.8888888888888888	81.015625	81015625.0
		0.7777777777777778	0.14984848484848484	7.0	[1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 1.]	0.8518518518518519	83.234375	83234375.0
		0.8888888888888888	0.021060606060606033	6.0	[0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0.]	0.9814814814814815	88.71875	88718750.0
		0.7962962962962963	0.13060606060606061	5.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1.]	0.8703703703703703	89.53125	89531250.0
		0.8148148148148148	0.0939393939393939	5.0	[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 1.]	0.9074074074074074	87.984375	87984375.0
		0.8333333333333334	0.11363636363636367	8.0	[0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 0. 1.]	0.8888888888888888	80.25	80250000.0
		0.82962962962963		5.6		0.89444444444444	85.434375	
	Tic-tac-toe	0.88020833333333	0.128593749999	9.0	[1. 1. 1. 1. 1. 1. 1. 1. 1.]	0.8802083333333334	371.015625	371015625.0

		34	99995					
		0.8958333333333334	0.10059027777777776	7.0	[1. 1. 1. 1. 1. 0. 1. 0. 1.]	0.90625	403.484375	403484375.0
		0.8802083333333334	0.12232638888888893	8.0	[1. 0. 1. 1. 1. 1. 1. 1. 1.]	0.8854166666666666	321.796875	321796875.0
		0.890625	0.11717013888888889	8.0	[1. 0. 1. 1. 1. 1. 1. 1. 1.]	0.890625	334.015625	334015625.0
		0.890625	0.11201388888888886	8.0	[1. 1. 1. 1. 1. 1. 1. 0. 1.]	0.8958333333333334	320.0625	320062500.0
		0.9010416666666666	0.10685763888888893	8.0	[1. 1. 1. 1. 1. 0. 1. 1. 1.]	0.9010416666666666	310.0625	310.0625
		0.9010416666666666	0.10796875000000003	9.0	[1. 1. 1. 1. 1. 1. 1. 1. 1.]	0.9010416666666666	315.4375	315437500.0
		0.90625	0.10281249999999999	9.0	[1. 1. 1. 1. 1. 1. 1. 1. 1.]	0.90625	325.78125	325781250.0
		0.859375	0.1326388888888889	8.0	[1. 0. 1. 1. 1. 1. 1. 1. 1.]	0.875	385.96875	385968750.0
		0.921875	0.08734375	9.0	[1. 1. 1. 1. 1. 1. 1. 1. 1.]	0.921875	361.9375	361937500.0
		0.89270833333333		8.3		0.89635416666667	344.95625	
	Vote	0.9	0.051375000000000046	3.0	[0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]	0.95	78.265625	78265625.0
		0.9666666666666667	0.01837500000000005	3.0	[0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]	0.9833333333333333	86.625	86625000.0
		0.9833333333333333	0.004375	7.0	[1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0.]	1.0	86.625	86625000.0
		0.9333333333333333	0.052000000000000046	4.0	[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0.]	0.95	79.40625	79406250.0
		1.0	0.001875	3.0	[0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0.]	1.0	77.453125	77453125.0
		0.9	0.03862499999999999	9.0	[0. 0. 0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1.]	0.9666666666666667	86.75	86750000.0
		0.9666666666666667	0.0025	4.0	[0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0.]	1.0	81.25	81250000.0
		0.9333333333333333	0.00625	10.0	[1. 0. 0. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0.]	1.0	92.0625	92062500.0
		0.9833333333333333	0.001875	3.0	[0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]	1.0	85.265625	85265625.0
		0.9666666666666667	0.001875	3.0	[0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]	1.0	81.578125	81578125.0
		0.95333333333333		4.9		0.985	83.528125	
	WaveformEW	0.871	0.1305	27.0	[0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1.	0.875	9044.796875	9044796875.0

					1. 0.]			
		0.88	0.11490999999999998	28.0	[1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1.]	0.891	9105.359375	9105359375.0
		0.873	0.13099	25.0	[0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 1. 1.]	0.874	9585.546875	9585.546875
		0.867	0.12853	31.0	[1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1.]	0.878	8940.0625	8940062500.0
		0.866	0.131	29.0	[1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1.]	0.875	8877.84375	89877843750.0
		0.86	0.12803	29.0	[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0.]	0.878	9364.59375	9364593750.0
		0.852	0.13644	27.0	[0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 0. 1. 1. 0. 1.]	0.869	9181.109375	9181109375.0
		0.878	0.11912999999999999	33	[0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0.]	0.888	8209.015625	8209015625.0
		0.863	0.14016	30	[1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 0. 1. 0. 0.]	0.866	8764.625	8764625000.0
		0.853	0.14337000000000003	27	[0. 0. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 1. 1. 0. 1. 1. 1.]	0.862	8494.140625	8494140625.0
		0.8663		28.6		0.8756	8956.709375	
	Wine	0.9666666666666667	0.001875	3.0	[0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]	1.0	101.578125	101578125.0
		0.5833333333333334	0.030576923076923 09	4.0	[0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0.]	0.9722222222222222	85.5625	85562500.0
		0.75	0.0038461538461538464	5.0	[1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0.]	1.0	89.46875	89468750.0
		0.66666666666666	0.057307692307	3.0	[1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0.]	0.9444444444444444	67.203125	67203125.0

		66	69233					
		0.7222222222222222	0.029038461538461548	2.0	[1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]	0.9722222222222222	70.125	70125000.0
		0.6666666666666666	0.05730769230769233	3.0	[1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]	0.9444444444444444	67.8125	67812500.0
		0.6388888888888888	0.05730769230769233	3.0	[1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0.]	0.9444444444444444	63.90625	63906250.0
		0.7222222222222222	0.05884615384615387	5.0	[1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0.]	0.9444444444444444	71.125	71125000.0
		0.7222222222222222	0.0038461538461538464	5.0	[1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]	1.0	72.53125	72531250.0
		0.6944444444444444	0.03211538461538463	6.0	[0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 0.]	0.9722222222222222	70.109375	70109375.0
		0.71333333333333		3.9		0.96944444444444	75.9421875	
	Zoo	0.95	0.004375	7.0	[0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1.]	1.0	62.25	62250000.0
		0.95	0.05262500000000005	5.0	[1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0.]	0.95	65.78125	65781250.0
		0.9	0.0025	4.0	[0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]	1.0	60.734375	60734375.0
		0.95	0.003125	5.0	[1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0.]	1.0	69.8125	69812500.0
		0.9	0.00375	6.0	[0. 1. 1. 0. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]	1.0	64.65625	64656250.0
		0.95	0.0025	4.0	[0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]	1.0	65.546875	65546875.0
		0.95	0.003125	5.0	[1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1. 0. 0.]	1.0	64.171875	64171875.0
		0.95	0.05262500000000005	5.0	[1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]	0.95	64.203125	64203125.0
		0.95	0.05262500000000005	5.0	[0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0.]	0.95	63.9375	63937500.0
		0.95	0.05262500000000005	5.0	[0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.]	0.95	65.515625	65515625.0
		0.94		5.1		0.98	64.6609375	

APPENDIX E

MA-HS-KNN EXPERIMENTAL RESULTS FOR ALL 18 UCI DATASETS

Algorithm	Dataset	Pre FS Accuracy	Fitness Score	Selected Features Count	Selected Feautes	Post FS Accuracy	Process time (s)	Process time (ms)
MA-HS-KNN	Exactly	0.73	0.06734615384615386	7	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]	0.985	304.640625	304640625.0
		0.705	0.05384615384615385	7	[1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0]	1.0	302.625	302625000.0
		0.74	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	301.84375	301843750.0
		0.705	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	291.28125	291281250.0
		0.76	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	297.71875	297718750.0
		0.715	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	286.9375	286937500.0
		0.74	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	325.421875	325421875.0
		0.695	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	307.703125	307703125.0
		0.68	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	295.453125	295453125.0
		0.795	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	303.59375	303593750.0
		0.72545454545455		6.2		0.9985	301.721875	
	Exactly2	0.72	0.2236923076923077	1	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.76	286.921875	286921875.0
		0.725	0.2236923076923077	1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]	0.76	278.015625	278015625.0
		0.735	0.2236923076923077	1	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]	0.76	295.3125	295312500.0
		0.73	0.2236923076923077	1	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]	0.76	281.640625	281640625.0
		0.705	0.2236923076923077	1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]	0.76	276.09375	276093750.0
		0.74	0.20176923076923073	4	[0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]	0.81	275.703125	275703125.0
		0.715	0.21846153846153843	5	[0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0]	0.8	284.21875	284218750.0
		0.74	0.2236923076923077	1	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.76	299.578125	299578125.0

		0.74	0.2236923076923077	1	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]	0.76	264.953125	264953125.0
		0.745	0.2236923076923077	1	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.76	275.765625	275765625.0
		0.7295		1.7		0.769	281.8203125	
	Breastcancer	0.6071428571428571	0.04571428571428573	2	[0, 1, 0, 0, 0, 0, 1, 0, 0, 0]	0.9714285714285714	218.515625	218515625.0
		0.5571428571428572	0.05571428571428572	3	[0, 1, 1, 0, 0, 0, 1, 0, 0, 0]	0.9714285714285714	219.828125	219828125.0
		0.6428571428571429	0.032857142857142814	2	[0, 0, 0, 1, 0, 0, 0, 1, 0, 0]	0.9857142857142858	219.609375	219609375.0
		0.6571428571428571	0.06214285714285713	3	[0, 1, 1, 0, 0, 0, 1, 0, 0, 0]	0.9642857142857143	222.125	222125000.0
		0.5928571428571429	0.05857142857142854	2	[0, 1, 0, 0, 1, 0, 0, 0, 0, 0]	0.9571428571428572	212.703125	212703125.0
		0.5642857142857143	0.04285714285714281	3	[0, 0, 1, 1, 0, 0, 0, 1, 0, 0]	0.9857142857142858	224.546875	224546875.0
		0.7	0.04928571428571432	3	[0, 1, 1, 0, 0, 0, 1, 0, 0, 0]	0.9785714285714285	215.8125	215812500.0
		0.5714285714285714	0.05214285714285714	2	[0, 0, 1, 0, 0, 0, 0, 1, 0, 0]	0.9642857142857143	246.984375	246984375.0
		0.55	0.02642857142857141	2	[0, 0, 1, 0, 0, 0, 1, 0, 0, 0]	0.9928571428571429	251.703125	251703125.0
		0.6428571428571429	0.032857142857142814	2	[0, 0, 1, 0, 0, 0, 1, 0, 0, 0]	0.9857142857142858	250.40625	250406250.0
		0.60857142857143		2.4		0.97571428571429	228.2234375	
	BreastEW	0.9210526315789473	0.06403508771929829	5	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]	0.9473684210526315	469.046875	469046875.0
		0.9122807017543859	0.04491228070175439	4	[0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]	0.9649122807017544	376.953125	376953125.0
		0.8596491228070176	0.056140350877193004	5	[0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.956140350877193	368.890625	368890625.0
		0.9122807017543859	0.054912280701754385	7	[0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]	0.9649122807017544	328.65625	328656250.0
		0.8859649122807017	0.0810526315789474	3	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]	0.9210526315789473	363.046875	363046875.0
		0.9298245614035088	0.04368421052631577	6	[1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]	0.9736842105263158	358.484375	358484375.0
		0.9122807017543859	0.03578947368421058	6	[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,	0.9824561403508771	411.484375	411484375.0

					0, 0]			
		0.9035087719298246	0.06736842105263163	6	[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]	0.9473684210526315	281.75	281750000.0
		0.8947368421052632	0.06280701754385967	7	[0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0]	0.956140350877193	420.546875	420546875.0
		0.9385964912280702	0.04245614035087725	8	[0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0]	0.9824561403508771	314.265625	314265625.0
		0.90701754385965		5.7		0.95964912280702	369.3125	
	CongressEW	0.896551724137931	0.06012931034482764	3	[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	0.9540229885057471	169.875	169875000.0
		0.9425287356321839	0.03728448275862065	1	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9655172413793104	167.625	167625000.0
		0.9310344827586207	0.035344827586206884	4	[0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]	0.9885057471264368	165.265625	165265625.0
		0.9425287356321839	0.04159482758620688	5	[0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1]	0.9885057471264368	168.59375	168593750.0
		0.9080459770114943	0.05603448275862065	4	[0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0]	0.9655172413793104	171.0625	171062500.0
		0.9195402298850575	0.025	4	[0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]	1.0	167.859375	167859375.0
		0.9310344827586207	0.02693965517241377	1	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9770114942528736	168.484375	168484375.0
		0.9770114942528736	0.03318965517241377	2	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]	0.9770114942528736	175.9375	175937500.0
		0.8850574712643678	0.07672413793103453	4	[0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	0.9425287356321839	165.328125	165328125.0
		0.9195402298850575	0.05797413793103452	1	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9425287356321839	165.953125	165953125.0
		0.92528735632184		2.9		0.97011494252874	168.5984375	
	HeartEW	0.7962962962962963	0.13974358974358975	3	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1]	0.8703703703703703	134.46875	134468750.0
		0.6666666666666666	0.1307692307692308	4	[0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]	0.8888888888888888	135.609375	135609375.0
		0.6851851851851852	0.14743589743589747	4	[0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1]	0.8703703703703703	140.59375	140593750.0
		0.6296296296296297	0.19743589743589748	4	[0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1]	0.8148148148148148	140.75	140750000.0
		0.7222222222222222	0.10512820512820513	5	[0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1]	0.9259259259259259	135.890625	135890625.0
		0.7037037037037037	0.18076923076923074	4	[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1]	0.8333333333333334	134.65625	134656250.0
		0.6666666666666666	0.1717948717948718	5	[0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1]	0.8518518518518519	141.609375	141609375.0

		0.6296296296296297	0.14743589743589747	4	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1]	0.8703703703703703	136.046875	136046875.0
		0.7962962962962963	0.09743589743589744	4	[0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]	0.9259259259259259	134.4375	134437500.0
		0.7222222222222222	0.10512820512820513	5	[0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1]	0.9259259259259259	138.0	138000000.0
		0.70185185185185		4.2		0.87777777777778	137.20625	
	IonosphereEW	0.8428571428571429	0.040420168067226904	5	[1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9714285714285714	325.625	325625000.0
		0.8428571428571429	0.09184873949579835	5	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]	0.9142857142857143	307.296875	307296875.0
		0.8571428571428571	0.07495798319327733	8	[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]	0.9428571428571428	301.53125	301531250.0
		0.8571428571428571	0.06319327731092439	4	[0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9428571428571428	302.21875	302218750.0
		0.8571428571428571	0.031596638655462195	2	[0, 0, 0, 0, 1, 0, 0, 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]	0.9714285714285714	298.046875	298046875.0
		0.8142857142857143	0.07899159663865545	5	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1]	0.9285714285714286	301.6875	301687500.0
		0.7857142857142857	0.0760504201680672	4	[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, 1, 0, 0, 0, 0, 0, 0, 0]	0.9285714285714286	302.390625	302390625.0
		0.8428571428571429	0.044453781512605005	2	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9571428571428572	276.890625	276890625.0
		0.8142857142857143	0.0760504201680672	4	[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9285714285714286	310.03125	310031250.0
		0.9	0.040420168067226904	5	[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]	0.9714285714285714	314.03125	314031250.0
		0.84142857142857		4.4		0.94571428571429	303.975	
	KrvskpEW	0.9624413145539906	0.06701877934272298	15	[1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0]	0.971830985915493	1292.609375	1292609375.0
		0.9561815336463224	0.07104851330203446	20	[1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0]	0.9827856025039123	1550.1875	1550187500.0
		0.9640062597809077	0.05989827856025037	17	[1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,	0.9859154929577465	1374.75	1374750000.0

					1, 0, 0, 0, 1, 1, 1, 1]			
		0.9655712050078247	0.08215962441314556	24	[1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1]	0.9827856025039123	1317.09375	1317093750.0
		0.9608763693270735	0.07683881064162751	16	[0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0]	0.9640062597809077	1264.515625	1264515625.0
		0.9655712050078247	0.06271517996870112	17	[1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1]	0.9827856025039123	1284.6875	1284687500.0
		0.9671361502347418	0.06271517996870112	17	[1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0]	0.9827856025039123	1310.375	1310375000.0
		0.9546165884194053	0.06283255086071993	14	[1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0]	0.97339593114241	1381.6875	1381687500.0
		0.9561815336463224	0.07120500782472612	16	[1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0]	0.9702660406885759	1418.1875	1418187500.0
		0.9608763693270735	0.07679968701095465	17	[1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0]	0.9671361502347418	1311.734375	1311734375.0
		0.96134585289515		17.3		0.97636932707355	1350.5828125	
	Lymphography	0.7333333333333333	0.05222222222222222	4	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1]	0.9666666666666667	113.046875	113046875.0
		0.6333333333333333	0.09333333333333332	6	[1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0]	0.9333333333333333	113.3125	113312500.0
		0.8	0.09333333333333332	6	[0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0]	0.9333333333333333	99.765625	99765625.0
		0.7666666666666667	0.08555555555555555	10	[0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0]	0.9666666666666667	115.015625	115015625.0
		0.7666666666666667	0.1422222222222222	4	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1]	0.8666666666666667	112.453125	112453125.0
		0.7666666666666667	0.09888888888888889	7	[0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1]	0.9333333333333333	116.921875	116921875.0
		0.8666666666666667	0.07444444444444444	8	[0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1]	0.9666666666666667	118.75	118750000.0
		0.7	0.10666666666666665	3	[0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]	0.9	119.34375	119343750.0
		0.8	0.09888888888888889	7	[0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]	0.9333333333333333	115.453125	115453125.0
		0.8333333333333334	0.08222222222222221	4	[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0]	0.9333333333333333	118.578125	118578125.0
		0.76666666666667		5.9		0.93333333333333	114.2640625	
	M-of-N	0.88	0.046153846153	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	306.28125	306281250.0

			846156					
		0.905	0.05384615384615385	7	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0]	1.0	307.96875	307968750.0
		0.86	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	306.03125	306031250.0
		0.905	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	304.34375	304343750.0
		0.925	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	311.015625	311015625.0
		0.895	0.05384615384615385	7	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0]	1.0	301.703125	301703125.0
		0.905	0.05384615384615385	7	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]	1.0	319.25	319250000.0
		0.85	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	308.34375	308343750.0
		0.875	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	299.9375	299937500.0
		0.895	0.05384615384615385	7	[1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0]	1.0	306.578125	306578125.0
		0.8895		6.4		1.0	307.1453125	
	PenglungEW	0.7333333333333333	0.023384615384615386	76	[1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]	1.0	135.390625	135390625.0
		0.8	0.027692307692307697	90	[0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,	1.0	129.0625	129062500.0

					0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0]			
		0.7333333333333333	0.08215384615384615	72	[0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]	0.9333333333333333	130.890625	130890625.0
		0.93333333333333	0.022461538461	73	[0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0,	1.0	137.875	137875000.0

		33	538463		0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 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, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]			
		1.0	0.024615384615 38462	80	[0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]	1.0	141.75	141750000.0

		0.6666666666666666	0.20430769230769227	79	[1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 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, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.8	125.5625	125562500.0
		0.8	0.08276923076923076	74	[1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0,	0.9333333333333333	129.296875	129296875.0

					1, 0, 0]			
		0.8	0.14123076923076922	69	[0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1]	0.8666666666666667	132.15625	132156250.0
		0.8	0.08615384615384615	85	[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,	0.9333333333333333	132.796875	132796875.0

					1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0]			
		0.6666666666666666	0.144	78	[0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]	0.8666666666666667	132.8125	132812500.0
		0.7933333333333333		77.6		0.9333333333333333	132.759375	
	Sonar	0.8333333333333334	0.09095238095238094	16	[1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0]	0.9285714285714286	115.5	115500000.0
		0.7380952380952381	0.11071428571428571	15	[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]	0.9047619047619048	160.703125	160703125.0
		0.7619047619047619	0.06952380952380957	16	[0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]	0.9523809523809523	163.40625	163406250.0
		0.8333333333333334	0.09095238095238094	16	[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]	0.9285714285714286	166.203125	166203125.0

		0.8333333333333334	0.08928571428571427	15	[1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9285714285714286	166.03125	166031250.0
		0.8333333333333334	0.053095238095238126	19	[0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1]	0.9761904761904762	162.53125	162531250.0
		0.7380952380952381	0.0876190476190476	14	[1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	0.9285714285714286	162.21875	162218750.0
		0.8809523809523809	0.05476190476190479	20	[1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0]	0.9761904761904762	165.59375	165593750.0
		0.8095238095238095	0.08928571428571427	15	[0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]	0.9285714285714286	164.453125	164453125.0
		0.8571428571428571	0.07452380952380958	19	[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]	0.9523809523809523	165.03125	165031250.0
				16.5		0.94047619047619	159.1671875	
	SpectEW	0.8518518518518519	0.12272727272727277	5	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0]	0.8888888888888888	125.453125	125453125.0
		0.7777777777777778	0.09393939393939393	6	[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1]	0.9259259259259259	105.078125	105078125.0
		0.8148148148148148	0.12727272727272732	6	[1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0]	0.8888888888888888	78.328125	78328125.0
		0.7592592592592593	0.1545454545454546	12	[1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0]	0.8888888888888888	103.09375	103093750.0
		0.7962962962962963	0.1363636363636364	8	[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1]	0.8888888888888888	109.703125	109703125.0
		0.7592592592592593	0.1818181818181818	7	[1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1]	0.8333333333333334	106.96875	106968750.0
		0.9259259259259259	0.06060606060606065	6	[0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0]	0.9629629629629629	109.46875	109468750.0
		0.74074074074074	0.160606060606	6	[0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,	0.8518518518518519	144.1875	144187500.0

		07	0606		0, 0, 0, 0, 1, 1, 0, 0]			
		0.77777777777777 78	0.106060606060 60602	5	[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1]	0.9074074074074074	139.84375	139843750.0
		0.77777777777777 78	0.166666666666 66669	11	[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1]	0.8703703703703703	135.640625	135640625.0
		0.80502645502646		7.2		0.89074074074074	115.7765625	
	Tic-tac-toe	0.85416666666666 66	0.214930555555 5556	5	[1, 1, 1, 0, 1, 0, 0, 0, 1]	0.8229166666666666	283.84375	283843750.0
		0.859375	0.200868055555 5556	5	[1, 0, 1, 0, 1, 0, 1, 0, 1]	0.8385416666666666	287.46875	287468750.0
		0.82291666666666 66	0.252430555555 55555	5	[1, 1, 1, 0, 1, 1, 0, 0, 0]	0.78125	284.78125	284781250.0
		0.80729166666666 66	0.227256944444 44447	4	[0, 1, 0, 0, 1, 0, 1, 1, 0]	0.796875	288.859375	288859375.0
		0.83333333333333 34	0.200868055555 5556	5	[1, 0, 0, 1, 1, 0, 1, 0, 1]	0.8385416666666666	303.65625	303656250.0
		0.83854166666666 66	0.224305555555 55556	5	[0, 1, 1, 0, 1, 1, 0, 0, 1]	0.8125	296.78125	296781250.0
		0.828125	0.200868055555 5556	5	[1, 0, 1, 1, 1, 0, 1, 0, 0]	0.8385416666666666	285.109375	285109375.0
		0.85416666666666 66	0.231250000000 00004	9	[1, 1, 1, 1, 1, 1, 1, 1, 1]	0.8541666666666666	295.140625	295140625.0
		0.80729166666666 66	0.219618055555 55552	5	[0, 0, 0, 1, 1, 0, 1, 1, 1]	0.8177083333333334	286.96875	286968750.0
		0.84895833333333 34	0.210243055555 55556	5	[1, 0, 0, 1, 1, 1, 0, 0, 1]	0.828125	296.390625	296390625.0
		0.83541666666667		5.3		0.82291666666667	290.9	
	Vote	0.93333333333333 33	0.033750000000 00005	3	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0]	0.9833333333333333	143.4375	143437500.0
		0.88333333333333 33	0.033750000000 00005	3	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0]	0.9833333333333333	138.609375	138609375.0
		0.96666666666666 67	0.03625	1	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9666666666666667	141.875	141875000.0
		0.88333333333333 33	0.040000000000 00005	4	[0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]	0.9833333333333333	139.671875	139671875.0
		0.91666666666666 66	0.018750000000 000003	3	[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]	1.0	137.859375	137859375.0
		0.86666666666666 67	0.063750000000 00004	3	[0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0]	0.95	146.75	146750000.0
		0.95	0.027500000000 000045	2	[0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9833333333333333	147.578125	147578125.0
		0.88333333333333 33	0.054999999999 99999	4	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0]	0.9666666666666667	140.59375	140593750.0
		0.96666666666666	0.018750000000	3	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,	1.0	147.5	147500000.0

		67	000003		1, 0]			
		0.9166666666666666	0.0125	2	[1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	1.0	152.140625	152140625.0
		0.91666666666667		2.8		0.98166666666667	143.6015625	
	WaveformEW	0.807	0.20970000000000005	18	[0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0]	0.817	5211.84375	5211843750.0
		0.824	0.18760000000000004	16	[0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1]	0.836	5166.046875	5166046875.0
		0.812	0.19190000000000004	17	[0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0	0.834	5190.25	5190250000.0
		0.818	0.20080000000000006	13	[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	0.813	4864.171875	4864171875.0
		0.804	0.20759999999999995	15	[0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	0.811	4816.515625	4816515625.0
		0.805	0.20160000000000006	18	[0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]	0.826	4880.734375	4880734375.0
		0.813	0.20080000000000003	22	[0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1]	0.838	5083.828125	5083828125.0
		0.791	0.19630000000000006	13	[0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	0.818	4820.4375	4820437500.0
		0.795	0.19410000000000005	15	[0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]	0.826	4957.390625	4957390625.0
		0.794	0.21070000000000005	22	[0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0]	0.827	4884.859375	4884859375.0
		0.8063		16.9		0.8246	4987.6078125	
	Wine	0.7222222222222222	0.03076923076923077	4	[1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0]	1.0	119.203125	119203125.0
		0.6944444444444444	0.03076923076923077	4	[0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]	1.0	115.46875	115468750.0
		0.75	0.038461538461538464	5	[1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0]	1.0	118.234375	118234375.0
		0.6944444444444444	0.05576923076923078	4	[1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0]	0.9722222222222222	119.234375	119234375.0
		0.7777777777777778	0.040384615384615394	2	[1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	0.9722222222222222	115.421875	115421875.0

		0.6944444444444444	0.04807692307692309	3	[1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]	0.9722222222222222	119.265625	119265625.0
		0.6388888888888888	0.03076923076923077	4	[1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0]	1.0	119.375	119375000.0
		0.75	0.05576923076923078	4	[1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0]	0.9722222222222222	115.53125	115531250.0
		0.7777777777777778	0.03076923076923077	4	[1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0]	1.0	118.109375	118109375.0
		0.6388888888888888	0.0653846153846154	2	[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]	0.9444444444444444	118.390625	118390625.0
		0.71388888888889		4.1		0.98333333333333	117.8234375	
	Zoo	0.85	0.07625000000000004	5	[0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]	0.95	107.90625	107906250.0
		0.8	0.10125000000000003	9	[0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1]	0.95	110.078125	110078125.0
		0.85	0.025	4	[0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0]	1.0	108.640625	108640625.0
		0.95	0.03125	5	[0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0]	1.0	105.578125	105578125.0
		0.75	0.043750000000000004	7	[0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0]	1.0	106.90625	106906250.0
		0.85	0.037500000000000006	6	[0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0]	1.0	131.34375	131343750.0
		0.9	0.037500000000000006	6	[1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0]	1.0	123.671875	123671875.0
		0.95	0.06375000000000004	3	[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]	0.95	127.4375	127437500.0
		0.9	0.06375000000000004	3	[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]	0.95	126.234375	126234375.0
		0.9	0.07000000000000003	4	[0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]	0.95	123.125	123125000.0
		0.87		5.2		0.975	123.125	

APPENDIX F

MA-HS-SVM EXPERIMENTAL RESULTS FOR ALL 18 UCI DATASETS

Algo	Dataset	iAcc	Gbest fitness	NUM	Selected feautes	fAcc	Process time (s)	Process time (ms)
MA-HS-SVM	Breastcancer	0.6571428571428571	0.04571428571428573	2	[0, 1, 0, 0, 0, 0, 1, 0, 0, 0]	0.9714285714285714	180.03125	180031250.0
		0.6571428571428571	0.05214285714285714	2	[0, 1, 1, 0, 0, 0, 0, 0, 0, 0]	0.9642857142857143	175.578125	165578125.0
		0.6571428571428571	0.03642857142857141	3.0	[0, 1, 1, 0, 0, 0, 0, 1, 0, 0]	0.9928571428571429	178.484375	178484375.0
		0.6571428571428571	0.05571428571428572	3	[0, 0, 1, 0, 0, 1, 0, 1, 0, 0]	0.9714285714285714	170.859375	170859375.0
		0.6571428571428571	0.06500000000000004	2	[0, 1, 0, 0, 1, 0, 0, 0, 0, 0]	0.95	174.96875	164968750.0
		0.6571428571428571	0.03642857142857141	3.0	[0, 0, 1, 1, 0, 0, 0, 1, 0, 0]	0.9928571428571429	180.140625	180140625.0
		0.6571428571428571	0.04928571428571432	3.0	[0, 0, 0, 1, 0, 0, 0, 1, 1, 0]	0.9785714285714285	179.03125	169031250.0
		0.6571428571428571	0.05214285714285714	2	[0, 0, 1, 0, 0, 0, 0, 1, 0, 0]	0.9642857142857143	170.59375	160593750.0
		0.6571428571428571	0.032857142857142814	2.0	[0, 1, 1, 0, 0, 0, 0, 0, 0, 0]	0.9857142857142858	174.46875	174468750.0
		0.6571428571428571	0.032857142857142814	2	[0, 0, 1, 0, 0, 0, 1, 0, 0, 0]	0.9857142857142858	171.109375	171109375.0
				2.4		0.97571428571429	175.5265625	
	BreastEW	0.06192982456140348	0.04442105263157897	2	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9385964912280702	171.6875	171687500.0
		0.9298245614035088	0.05280701754385968	4.0	[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.956140350877193	176.75	176750000.0
		0.8508771929824561	0.060701754385964965	4	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9473684210526315	209.796875	209796875.0
		0.9122807017543859	0.06526315789473681	3.0	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9385964912280702	184.9375	184937500.0
		0.8771929824561403	0.08771929824561406	5.0	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]	0.9210526315789473	163.59375	163593750.0

		0.9298245614035088	0.0470175438596491	7	[0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9736842105263158	165.109375	165109375.0
		0.9210526315789473	0.04035087719298243	5	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0]	0.9736842105263158	171.625	171625000.0
		0.8947368421052632	0.07982456140350877	5	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0]	0.9298245614035088	165.140625	165140625.0
		0.9210526315789473	0.05157894736842106	6	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1]	0.9649122807017544	178.234375	178234375.0
		0.9298245614035088	0.05157894736842106	6.0	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1]	0.9649122807017544	163.28125	163281250.0
				4.7		0.95087719298246	175.015625	
	CongressEW	0.9310344827586207	0.06012931034482764	3.0	[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	0.9540229885057471	128.796875	128796875.0
		0.9655172413793104	0.03728448275862065	1.0	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9655172413793104	124.890625	124890625.0
		0.9655172413793104	0.03943965517241377	3.0	[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	0.9770114942528736	121.265625	121265625.0
		0.9770114942528736	0.03728448275862065	1.0	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9655172413793104	125.515625	125515625.0
		0.9425287356321839	0.06831896551724141	1.0	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9310344827586207	124.28125	124281250.0
		0.9770114942528736	0.018750000000000003	3.0	[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	1.0	126.328125	126328125.0
		0.9770114942528736	0.02693965517241377	1.0	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9770114942528736	121.328125	121328125.0
		0.9770114942528736	0.03728448275862065	1.0	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9655172413793104	127.234375	127234375.0
		0.9080459770114943	0.07672413793103453	4.0	[0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0]	0.9425287356321839	118.65625	118656250.0
		0.9425287356321839	0.05603448275862065	4.0	[0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0]	0.9655172413793104	122.828125	122828125.0
				2.2		0.96436781609195	124.1125	
	Exactly	0.735	0.06415384615384617	6.0	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	0.98	563.65625	563656250.0
		0.7	0.2190384615384616	8.0	[1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0]	0.825	527.375	527375000.0
		0.71	0.12134615384615381	7.0	[1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0]	0.925	523.828125	523828125.0
		0.7	0.2100384615384616	8.0	[1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0]	0.835	484.234375	484234375.0
		0.735	0.073153846153	6.0	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	0.97	561.765625	561765625.0

			84618					
		0.725	0.18303846153846154	8.0	[1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0]	0.865	459.609375	459609375.0
		0.73	0.09884615384615389	7	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]	0.95	519.703125	519703125.0
		0.695	0.2145384615384616	8	[1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0]	0.83	477.0	477000000.0
		0.67	0.2100384615384616	8	[1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0]	0.835	475.90625	475906250.0
		0.72	0.07315384615384618	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	0.97	567.1875	567187500.0
				7.2		0.8985	516.0265625	
	Exactly2	0.765	0.2236923076923077	1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]	0.76	418.28125	418281250.0
		0.76	0.2236923076923077	1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]	0.76	412.15625	412156250.0
		0.765	0.2236923076923077	1	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]	0.76	410.796875	410796875.0
		0.755	0.2236923076923077	1	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.76	414.828125	414828125.0
		0.765	0.2236923076923077	1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]	0.76	411.84375	411843750.0
		0.76	0.2236923076923077	1	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.76	412.640625	412640625.0
		0.765	0.2236923076923077	1	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.76	404.546875	404546875.0
		0.75	0.2236923076923077	1	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.76	393.109375	393109375.0
		0.76	0.2236923076923077	1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]	0.76	382.265625	382265625.0
		0.765	0.2236923076923077	1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]	0.76	381.0	381000000.0
				1		0.76	404.146875	
	HeartEW	0.7407407407407407	0.14743589743589747	4.0	[0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1]	0.8703703703703703	115.546875	115546875.0
		0.6296296296296297	0.13846153846153852	5.0	[0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0]	0.8888888888888888	112.609375	112609375.0
		0.6851851851851852	0.12307692307692313	3.0	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0]	0.8888888888888888	113.6875	113687500.0
		0.6481481481481481	0.18076923076923074	4.0	[0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0]	0.8333333333333334	118.734375	118734375.0
		0.6666666666666666	0.11282051282051282	6	[0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1]	0.9259259259259259	116.890625	116890625.0
		0.7222222222222222	0.18717948717948718	7	[0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0]	0.8518518518518519	118.375	118375000.0
		0.57407407407407	0.139743589743	3	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]	0.8703703703703703	116.171875	116171875.0

		41	58975					
		0.7037037037037037	0.1538461538461539	7.0	[0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1]	0.8888888888888888	115.84375	115843750.0
		0.6481481481481481	0.09615384615384617	6.0	[0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0]	0.9444444444444444	112.109375	112109375.0
		0.5555555555555556	0.09743589743589744	4.0	[0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]	0.9259259259259259	113.78125	113781250.0
				4.9		0.88888888888889	115.375	
	Ionosphere	0.9857142857142858	0.02647058823529412	9	[1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0]	1.0	126.40625	126406250.0
		0.9285714285714286	0.062100840336134416	8.0	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]	0.9571428571428572	120.21875	120218750.0
		0.9428571428571428	0.03638655462184869	8	[0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0]	0.9857142857142858	121.828125	121828125.0
		0.9142857142857143	0.062100840336134416	8	[0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]	0.9571428571428572	125.234375	125234375.0
		0.9428571428571428	0.03344537815126046	7	[0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]	0.9857142857142858	125.75	125750000.0
		0.9142857142857143	0.0679831932773109	10	[0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1]	0.9571428571428572	126.4375	126437500.0
		0.9571428571428572	0.04226890756302517	10	[0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0]	0.9857142857142858	126.453125	126453125.0
		0.9571428571428572	0.03235294117647059	11	[0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1]	1.0	126.078125	126078125.0
		0.9285714285714286	0.04226890756302517	10	[0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0]	0.9857142857142858	122.125	122125000.0
		0.9714285714285714	0.023529411764705882	8	[0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	1.0	126.59375	126593750.0
				8.9		0.98142857142857	124.7125	
	KrVsKpEW	0.9765258215962441	0.05563380281690139	18	[1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0]	0.9937402190923318	3746.359375	3746359375.0
		0.974960876369327	0.061306729264475744	17	[1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1]	0.9843505477308294	3593.75	3593750000.0
		0.97496087636932	0.061306729264	17	[1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0,	0.9843505477308294	3676.703125	3676703125.0

		7	475744		1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0]			
		0.9640062597809077	0.07402190923317686	16	[1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0]	0.9671361502347418	3670.484375	3670484375.0
		0.97339593114241	0.058646322378716774	13	[1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]	0.974960876369327	3856.015625	3856015625.0
		0.9843505477308294	0.060172143974960846	10	[1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0]	0.9640062597809077	3824.125	3824125000.0
		0.9812206572769953	0.05301251956181538	13	[1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]	0.9812206572769953	3566.6875	3566687500.0
		0.9780907668231612	0.058568075117370944	15	[1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1]	0.9812206572769953	3459.46875	3459468750.0
		0.9702660406885759	0.06287167449139279	13	[1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]	0.9702660406885759	3513.046875	3513046875.0
		0.9796557120500783	0.06416275430359936	16	[1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0]	0.9780907668231612	3404.53125	3404531250.0
				14.8		0.97783818466354	3631.1171875	
	Lymphography	0.8333333333333334	0.06333333333333332	6	[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1]	0.9666666666666667	155.21875	155218750.0
		0.6	0.10999999999999999	9	[0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0]	0.9333333333333333	152.703125	152703125.0
		0.8	0.10666666666666665	3	[0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]	0.9	156.71875	156718750.0
		0.8	0.09333333333333332	6	[0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0]	0.9333333333333333	159.234375	159234375.0
		0.7666666666666667	0.13666666666666666	3	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]	0.8666666666666667	162.546875	162546875.0
		0.8	0.11777777777777776	5	[0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]	0.9	148.0625	148062500.0
		0.8666666666666667	0.12333333333333332	6	[0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0]	0.9	157.859375	157859375.0
		0.7666666666666667	0.10444444444444444	8	[0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0]	0.9333333333333333	161.0	161000000.0
		0.8333333333333334	0.09888888888888889	7	[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1]	0.9333333333333333	162.375	162375000.0
		0.7666666666666667	0.08777777777777777	5	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0]	0.9333333333333333	160.203125	160203125.0
				5.8		0.92	157.5921875	

	M-of-n	1.0	0.05384615384615385	7	[1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	517.75	517750000.0
		1.0	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	431.78125	431781250.0
		1.0	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	380.5625	380562500.0
		1.0	0.05384615384615385	7	[1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0]	1.0	521.234375	521234375.0
		1.0	0.046153846153846156	6.0	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	445.5	445500000.0
		1.0	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	597.4375	597437500.0
		1.0	0.05384615384615385	7	[1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0]	1.0	563.015625	563015625.0
		1.0	0.046153846153846156	6.0	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	499.0	499000000.0
		1.0	0.05384615384615385	7.0	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]	1.0	525.4375	525437500.0
		1.0	0.046153846153846156	6	[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0]	1.0	736.140625	736140625.0
				6.4		1.0	521.7859375	
	PenglungEW	0.8	0.13876923076923076	61	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 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, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.8666666666666667	209.265625	209265625.0

		0.7333333333333333	0.1409230769230769	68	[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]	0.8666666666666667	199.140625	199140625.0
		0.8	0.03169230769230769	103	[0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,	1.0	209.046875	209046875.0

					1, 1, 1, 0]				
		0.8666666666666667	0.0803076923076923	66	[0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 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, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0]	0.9333333333333333	215.1875	215187500.0
		0.8	0.08307692307692308	75	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,	0.9333333333333333	192.578125	192578125.0

					0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]			
		0.6	0.26123076923076927	69	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0]	0.7333333333333333	198.46875	198468750.0
		0.8	0.08215384615384615	72	[0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,	0.9333333333333333	205.90625	205906250.0

					0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]			
		0.6666666666666666	0.2052307692307692	82	[1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]	0.8	198.09375	198093750.0
		0.8	0.09292307692307691	107	[0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,	0.9333333333333333	202.640625	202640625.0

					1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]			
		0.6666666666666666	0.1972307692307692	56	[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0]	0.8	210.859375	210859375.0
				75.9		0.88	204.11875	
	Sonar	0.7857142857142857	0.10595238095238094	25	[0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1]	0.9285714285714286	192.90625	192906250.0
		0.8095238095238095	0.14880952380952384	25	[0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0]	0.8809523809523809	202.640625	202640625.0
		0.8571428571428571	0.03833333333333334	23	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1]	1.0	204.015625	204015625.0
		0.7857142857142857	0.07619047619047624	20	[0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,	0.9523809523809523	191.21875	191218750.0

					0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1]			
		0.9047619047619048	0.07619047619047624	20	[1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1]	0.9523809523809523	199.796875	199796875.0
		0.8333333333333334	0.10119047619047625	35	[1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1]	0.9523809523809523	183.015625	183015625.0
		0.8095238095238095	0.11071428571428571	15	[0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9047619047619048	166.640625	166640625.0
		0.8809523809523809	0.08619047619047623	26	[1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0]	0.9523809523809523	148.703125	148703125.0
		0.8333333333333334	0.10571428571428572	12	[0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1]	0.9047619047619048	153.8125	153812500.0
		0.7857142857142857	0.11071428571428571	15	[1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]	0.9047619047619048	147.890625	147890625.0
				21.6		0.93333333333333	179.0640625	
	SpectEW	0.8703703703703703	0.1363636363636364	8	[1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1]	0.8888888888888888	76.0625	76062500.0
		0.8703703703703703	0.10303030303030303	8	[0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1]	0.9259259259259259	92.078125	92078125.0
		0.8148148148148148	0.12727272727272732	6	[1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0]	0.8888888888888888	88.9375	88937500.0
		0.8148148148148148	0.16515151515151516	7	[0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0]	0.8518518518518519	88.875	88875000.0
		0.8148148148148148	0.12727272727272732	6.0	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0]	0.8888888888888888	103.359375	103359375.0
		0.7777777777777778	0.1727272727272727	5	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1]	0.8333333333333334	86.015625	86015625.0
		0.88888888888888	0.069696969696	8	[1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0,	0.9629629629629629	91.109375	91109375.0

		88	96976		0, 0, 1, 0, 0, 0, 0, 1, 0]			
		0.7962962962962963	0.1484848484848485	7	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1]	0.8703703703703703	85.265625	85265625.0
		0.8148148148148148	0.12272727272727277	5	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1]	0.8888888888888888	106.421875	106421875.0
		0.8333333333333334	0.15606060606060607	5	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1]	0.8518518518518519	87.34375	87343750.0
				6.5		0.88518518518519	90.546875	
	Tic-tac-toe	0.8802083333333334	0.19496527777777783	7	[1, 0, 1, 1, 1, 0, 1, 1, 1]	0.8697916666666666	396.015625	396015625.0
		0.8958333333333334	0.16215277777777778	7.0	[1, 1, 1, 1, 1, 0, 1, 0, 1]	0.90625	422.375	422375000.0
		0.8802083333333334	0.19201388888888893	8.0	[1, 0, 1, 1, 1, 1, 1, 1, 1]	0.8854166666666666	426.21875	426218750.0
		0.890625	0.1809027777777778	7	[1, 0, 1, 0, 1, 1, 1, 1, 1]	0.8854166666666666	393.0	393000000.0
		0.890625	0.18263888888888885	8.0	[1, 1, 1, 1, 1, 1, 1, 0, 1]	0.8958333333333334	414.09375	414093750.0
		0.9010416666666666	0.17795138888888892	8.0	[1, 1, 1, 1, 1, 0, 1, 1, 1]	0.9010416666666666	401.109375	401109375.0
		0.9010416666666666	0.1809027777777778	7	[1, 1, 1, 1, 1, 0, 1, 0, 1]	0.8854166666666666	498.09375	498093750.0
		0.90625	0.184375	9.0	[1, 1, 1, 1, 1, 1, 1, 1, 1]	0.90625	443.40625	443406250.0
		0.859375	0.19965277777777773	7	[1, 0, 1, 0, 1, 1, 1, 1, 1]	0.8645833333333334	399.53125	399531250.0
		0.921875	0.1703125	9.0	[1, 1, 1, 1, 1, 1, 1, 1, 1]	0.921875	415.53125	415531250.0
				7.7		0.8921875	420.9375	
	Vote	0.9	0.06375000000000004	3.0	[0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.95	84.3125	84312500.0
		0.9666666666666667	0.04625000000000005	5	[0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0]	0.9833333333333333	102.65625	102656250.0
		0.9833333333333333	0.04000000000000005	4	[0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9833333333333333	107.328125	107328125.0
		0.9333333333333333	0.07874999999999999	3	[0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0]	0.9333333333333333	85.0625	85062500.0
		1.0	0.018750000000000003	3.0	[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]	1.0	105.21875	105218750.0
		0.9	0.06375000000000004	3	[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0]	0.95	83.890625	83890625.0
		0.9666666666666667	0.027500000000000045	2	[0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]	0.9833333333333333	81.890625	81890625.0
		0.9333333333333333	0.04875	3	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0]	0.9666666666666667	104.9375	104937500.0
		0.98333333333333	0.021250000000	1	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,	0.9833333333333333	83.890625	83890625.0

		33	000047		0, 0, 0]			
		0.96666666666666 67	0.021250000000 000047	1	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0.9833333333333333	104.921875	104921875.0
				2.8		0.97166666666667	94.4109375	
	WaveformEW	0.871	0.173700000000 00002	18	[0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0]	0.857	9303.84375	9303843750.0
		0.88	0.157899999999 99998	25	[1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1]	0.894	9539.203125	9539203125.0
		0.873	0.166000000000 00004	16	[1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0]	0.86	8867.28125	8867281250.0
		0.867	0.169800000000 00003	15	[1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]	0.853	12263.640625	12263640625.0
		0.866	0.168600000000 00003	21	[0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0]	0.871	9239.625	9239625000.0
		0.86	0.1769	20	[0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0]	0.859	8993.859375	8993859375.0
		0.852	0.177100000000 00004	19	[1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0]	0.856	8772.046875	8772046875.0
		0.878	0.1643	20	[0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1]	0.873	8673.78125	8673781250.0
		0.863	0.179200000000 00003	22	[0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1]	0.862	10130.453125	10130453125.0
		0.853	0.181800000000 00004	18	[0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0]	0.848	9914.625	9914625000.0
				19.4		0.8633	9569.8359375	
	Wine	0.66666666666666 66	0.048076923076 92309	3.0	[1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	0.9722222222222222	76.390625	76390625.0
		0.58333333333333 34	0.063461538461 53847	5	[0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0]	0.9722222222222222	95.921875	95921875.0
		0.75	0.040384615384 615394	2	[1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	0.9722222222222222	79.671875	79671875.0
		0.66666666666666 66	0.073076923076 9231	3.0	[1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]	0.9444444444444444	76.484375	76484375.0
		0.72222222222222 22	0.040384615384 615394	2.0	[1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	0.9722222222222222	81.828125	81828125.0
		0.66666666666666	0.073076923076	3.0	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]	0.9444444444444444	76.015625	76015625.0

		66	9231					
		0.63888888888888888	0.0730769230769231	3.0	[0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]	0.9444444444444444	81.390625	81390625.0
		0.7222222222222222	0.09038461538461542	2	[1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	0.9166666666666666	97.640625	97640625.0
		0.7222222222222222	0.038461538461538464	5.0	[1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0]	1.0	98.296875	98296875.0
		0.6944444444444444	0.06346153846153847	5	[0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0]	0.9722222222222222	97.15625	97156250.0
				3.3		0.96111111111111	86.0796875	
	Zoo	0.95	0.043750000000000004	7.0	[0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0]	1.0	96.46875	96468750.0
		0.95	0.08875000000000005	7	[0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0]	0.95	73.1875	73187500.0
		0.9	0.03125	5	[0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0]	1.0	74.84375	74843750.0
		0.95	0.037500000000000006	6	[0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0]	1.0	81.078125	81078125.0
		0.9	0.037500000000000006	6.0	[0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0]	1.0	75.53125	75531250.0
		0.95	0.025	4.0	[0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0]	1.0	74.65625	74656250.0
		0.95	0.03125	5.0	[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0]	1.0	94.015625	94015625.0
		0.95	0.08250000000000005	6	[0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0]	0.95	88.34375	88343750.0
		0.95	0.08250000000000005	6	[1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0]	0.95	73.296875	73296875.0
		0.95	0.08250000000000005	6	[0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0]	0.95	74.546875	74546875.0
				5.8		0.98	80.596875	

APPENDIX G

CURRICULUM VITAE

GARY LOUIS P. GARCIA
Km. 68, Camanchiles, Matanao, Davao del Sur
Contact No.: 09513784733
Email Address: garylouis.p.garcia@gmail.com

PERSONAL INFORMATION

Date of Birth: June 28, 2002
Place of Birth: Quezon St., Lupon, Davao Oriental
Age: 21
Nationality: Filipino
Religion: Seventh-day Adventist
Civil Status: Single

EDUCATIONAL BACKGROUND

Tertiary School: SOUTH PHILIPPINE ADVENTIST COLLEGE
Bachelor of Science in Computer Science, 2020-2024
Km. 68, Camanchiles, Matanao, Davao del Sur

Senior High School: SOUTH PHILIPPINE ADVENTIST COLLEGE
Science, Technology, Engineering, and Mathematics Str
Km. 68, Camanchiles, Matanao, Davao del Sur

Junior High School: PALAWAN ADVENTIST ACADEMY
Tacras, Narra, Palawan

Elementary School: PALAWAN ADVENTIST ELEMENTARY SCHOOL
Tacras, Narra, Palawan