Parallelization of BioInspired Computing algorithms using MASS JAVA

Venkata Ramani Srilekha Bandaru

A whitepaper submitted in partial fulfillment of

the requirements for the degree of

Master of Science in Computer Science and Software Engineering

University of Washington, Bothell

June 05th 2023

Project Committee:

Professor Munehiro Fukuda, Committee Chair

Professor Michael Stiber, Committee Member

Professor Min Chen, Committee Member

Contents

Listings

List of Figures

List of Tables

Abstract

The exponential growth of big data has posed significant challenges for traditional optimization algorithms in effectively processing and extracting meaningful insights from large-scale datasets. In this context, bio-inspired computing has emerged as a promising approach, drawing inspiration from natural systems and phenomena. By mimicking biological processes such as evolution, swarm behavior, and natural selection, bio-inspired algorithms offer innovative solutions for optimizing data processing, pattern recognition, classification, clustering, and other tasks related to big data analytics.

Parallelizing bio-inspired computing algorithms is crucial for achieving improved performance and scalability. This accelerates the optimization process and enhances the efficiency of solving challenging problems. Multi-Agent Spatial Simulation (MASS) is an agent-based modeling library that has been used in great extent to parallelize a variety of simulations and data analysis applications. Building on this foundation, the implementation of Bio-inspired Computing algorithms project is an exploration into the advantages of using MASS Java to parallelize computationally complex algorithms [1].

This project presents the applications of algorithm designs for agent-based versions of Swarm Based Computation, Evolutionary Computation and Ecological Computation Algorithms. In addition to the designs of the algorithms, we present an analysis of programmability and performance comparing MASS Java to another agent based modeling framework named Repast Simphony.

1 Introduction

1.1 Background

Bio-inspired computing is a branch of computer science and engineering that takes inspiration from biological systems to develop algorithms and systems to solve complex problems. The underlying principle of bio-inspired computing is that nature has evolved efficient and effective solutions to many problems, and by studying these solutions, we can develop better algorithms and systems.

Bio-inspired computing draws inspiration from a wide range of biological systems, including genetic algorithms that mimic the process of natural selection, neural networks that simulate the structure and function of the brain, swarm intelligence that studies the collective behavior of social animals like bees and ants, and artificial immune systems that mimic the immune response of living organisms.

One of the key advantages of bio-inspired computing is its ability to handle complex, nonlinear problems that are difficult to solve using traditional methods. This makes bio-inspired algorithms particularly useful in fields like big data analytics, where large volumes of data can be analyzed to extract patterns and insights that are not immediately apparent.

For example, bio-inspired computing can be used to develop algorithms for data clustering, where data points are grouped together based on their similarities. This can help identify patterns in data that might be useful for making predictions or developing new products.

Another example is in the field of image processing, where bio-inspired algorithms can be used to simulate the process of human vision, and extract features from images that are useful for classification and recognition.

Another example is in the field of manufacturing processes, where bio-inspired algorithms in Job Shop Scheduling Problem enable efficient exploration of job-machine assignments, considering complex constraints and dynamic factors in manufacturing processes. They optimize scheduling

outcomes, reduce makespan, and enhance resource utilization, leading to improved efficiency and productivity.

In order to improve the speed and accuracy of solving complex problems, biological optimization algorithms harness the behavioral patterns of animals and humans. Although several types of optimization techniques exist, they can be broadly categorized into two fundamental steps [2]:

Exploration: This involves the process of agents traversing through data to identify an acceptable target.
Exploitation: Here, agents leverage the knowledge gained through exploration to draw others to their position.

Bio-inspired algorithms for big data analytics are classified into 3 categories :

- Swarm Based Computation algorithms
- Evolutionary Computation algorithms
- Ecological Computation algorithms

1.2 Motivation

In recent times, the computational complexity of various decision making processes has become increasingly reliant on the availability of high-quality data, commonly referred to as big data. These data sets are characterized by their voluminous, complex, and ever-expanding nature, rendering traditional data processing software is inadequate in managing them. As a result, there is an increasing need for computationally efficient techniques that can cope with the challenges posed by big data. Bio-inspired computation provides a viable solution, as it encompasses biologically-inspired processes that harness the adaptability and self-learning capabilities of biological systems to address complex problems. This approach shows great promise in addressing issues related to scalability, task distribution, fault tolerance, and security challenges in big data computing environments. [3].

The motivation for solving bio-inspired algorithms in parallel setting is to address the limitations of sequential processing that hinder the scalability and efficiency of these algorithms. In the context of big data computing, parallel processing can significantly reduce the time required for processing large data sets, increase the processing power of the algorithm, and enable the use of large-scale distributed computing systems. Additionally, parallel processing can improve load balancing, and resource utilization, making the algorithms more robust and efficient. Therefore, exploring parallel implementation of bio-inspired algorithms can lead to significant advancements in solving complex problems and addressing challenges in diverse fields such as Image Processing, Optimization, and Machine Learning.

1.3 Goal

The search capabilities of biological optimizations like Exploring and Exploiting, best suite the nature of Agent Based Modeling. So we will be implementing 3 algorithms one from each category of bio-inspired computing and implement their applications using Multi Agent Spatial Simulation (MASS) library and another open source Agent based Modeling framework called Repast Simphony.
First we will be implementing an application of the algorithm in sequential setting and then we will be parallelizing it using MASS and Repast Simphony.

1. **Design Agent based approaches :** The primary goal of the project is to design and implement agent based algorithms for :
    - Multi modal Image Segmentation using Artificial Bee Colony.
    - Building Agent Based Classification System using Genetic Algorithm.
    - Job Shop Scheduling Problem using Bio-geography based Optimization algorithm.
2. **Performance Measurements :** To measure the performance of above 3 applications in sequential, MASS and Repast Simphony.
3. **Programmability Analysis :** Examine the programmability of the applications in MASS and compare it with sequential and Repast Simphony.

2 Libraries for Agent Based Modeling

2.1 Multi Agent Spatial Simulation(MASS)

This library utilizes Agent-Based Modeling (ABM) to do parallel computation over a cluster of nodes. It provides an innate programming framework to do big data processing and analysis. The MASS library contains two important classes called Places and Agents. Places is a distributed matrix of simulated spaces over a cluster of computing nodes. Each place can store information which can be pointed to by a set of network independent matrix indices, Places are fixed spaces and cant be moved after initialization whereas Agents are a set of execution instances which can migrate from place to place and also carry information to and fro from a place, by putting information on a place one agent can help other agent by providing information which was collected till then. MASS contains 2 types of agents called Static Agents and Dynamic Agents. Static Agents are the agents which perform the computations and store it on the Place but do not migrate or create more agents, whereas Dynamic Agents are the agents which migrate from place to place and spawn children from themselves to increase the agent population. MASS allows the user to customize the behavior of Places and Agents. [4]

2.2 Repast Simphony

Repast Simphony is an open-source agent-based modeling and simulation platform. It provides a framework for building and executing agent-based simulations, which are used to model complex systems made up of individual ”agents” that interact with each other and their environment. [5].It provides a set of tools and libraries for building and running agent-based simulations, including a graphical user interface for designing and running simulations, a Java-based simulation engine, and a set of pre-built simulation models and templates.

For benchmarking the performance of the MASS library, We are considering using Repast Simphony and compare their performance and scalability measurements.

3 Classes of Bio Inspired Computing Algorithms

3.1 Swarm Based Computation

Swarm-based computation is a type of bio-inspired computation algorithm that takes inspiration from the behavior of social organisms, such as ants, bees, and birds, to solve complex optimization problems. The basic idea is to simulate the collective behavior of a swarm of individuals in finding the optimal solution for a given problem.
There are several types of swarm-based computation algorithms, such as:

- Particle Swarm Optimization (PSO)
- Ant Colony Optimization (ACO)
- Artificial Bee Colony Optimization (ABC)

Each algorithm has its own unique characteristics, but they all share the same basic principle of using a population of agents, or particles, to search for the optimal solution.

In ABC, the algorithm simulates the foraging behavior of bees in finding the best food source. The bees communicate with each other through a dance language, where the direction and intensity of the dance convey information about the quality and location of the food source. The algorithm uses this information to update the position of the bees in the search space and converge to the optimal solution.

We have worked on implementing Artificial Bee colony algorithm to implement Multi Threshold Image Segmentation.
Image segmentation is the process of dividing an image into multiple regions or segments, each of which corresponds to a distinct object or part of an image. The goal of image segmentation is to simplify or change the representation of an image into something more meaningful and easier to analyze.
The process involves grouping pixels in an image into clusters based on their visual properties

such as color, texture, intensity, or other features. The result is a set of segments that can be used to isolate and analyze specific objects or regions within an image.

The main reason why we chose to implement Image segmentation with ABC is because when I read this paper [6], it was one of the first to mention the different types of methods that can be used for image segmentation. One of these methods is called *edge-line oriented detection*, which involves detecting edges and connecting them to form a line segment, and then connecting multiple line segments to form an object. However, this method faced the problem of combining line segments to form coherent regions that could be used in the future.

Another method for image segmentation is called *region growing*, which involves dividing the image into basic regions and then merging adjacent regions based on their similarity. However, this method faced the challenge of selecting the initial regions and merging them together, as well as potentially missing some clarity from the original image.

The final method mentioned in the paper is *Histogram based image segmentation*, for threshold setting, which has been one of the most successful methodologies. This approach involves computing a histogram for the image intensity values and analyzing it to determine a threshold setting for separating an object from the background but after doing some research the following paper [6] suggested that while performing image segmentation using histogram most of the time was consumed during the histogram generation and processing. Though Image Segmentation can be done in multiple ways *Thresholding* is one simple but effective tool which helps in isolating the objects of interest. *Threshold* selection can be done in 2 ways bi-level and multi-level, multiple objects in single scene can be detected using multiple threshold values. But implementing Image segmentation using multi level thresholding can cause some inconveniences (i). They might not have a semantic solution when number of objects to be detected increases. (ii). They may show slow convergence or high computational costs.

Usually, image processing tasks are numerically intensive, and therefore, a meta-heuristic algorithm such as Artificial Bee Colony, presented in [7], is a viable option for solving numerical optimization problems. Thus, We decided to implement Artificial Bee Colony to compute Multi-level Threshold selection for Image Segmentation using Gaussian approximations and histogram-based methods.

Artificial Bee Colony Explanation

A Bee colony usually contains worker bees and queen bee and all the bees follow a swarm based technique to collect required resources(ex: nectar, pollen etc). In the Artificial bee colony we have 3 types of bees namely Employee Bees, Onlooker Bees and Scout Bees/ Unemployed Bee. All these bees mainly follow two processes called *"Foraging"* and *"Recruitment"*.

Foraging: is a process in which Employee bees go around to find the food resources and if they find any potential resource then they will do a waggle dance to let the Unemployed bees know that they have found a potential resource and will let the unemployed bees know about the location of the resource.

Recruitment: the process where the unemployed bee finds the resource and gets recruited as an employee bee is known as Recruitment.

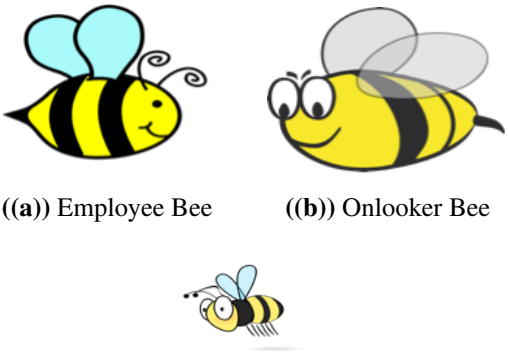
The *"Foraging"* and *"Recruitment"* processes in ABC allow the artificial bees to explore the search space and share information about the quality of the solutions found by each bee. This allows the swarm to collectively converge towards better solutions over time, similar to how a colony of honeybees collaborates to find the best food sources in their environment..

We will be applying this algorithm for Region based thresholding in Image Segmentation [8]. The idea of applying this algorithm for Multi Level Image Segmentation is because segmenting high quality images is a tedious process and takes a lot of time. In normal traditional Image segmentation algorithms for Multi level thresholding, the algorithm uses exhausted searching methods that expend too much computational time. So by applying Artificial Bee Colony algorithm for Image Segmentation, the algorithm will find a potential search space and employ multiple individuals which means ABC can achieve fast computational ability than traditional algorithms. ABC performs better than other algorithms because it has a special characteristic of strong global search ability i.e., one dimension search strategy where bees in ABC, search for potential solution one by one dimension.By parallellizing this ABC we are looking forward to achieve segmented output in lesser amount of time.

As part of our project, for this application we am planning on implementing an Artificial Bee Colony algorithm for 16 bit high quality images. This Segmentation will be useful in medical, astronomical and gaming fields by detecting objects in image.

Artificial Bee Colony Algorithm

Representation of Bees



((a)) Employee Bee **((b))** Onlooker Bee

((c)) Scout Bee

Figure 1: Representations of the bees used in this documentation.

If we observe Fig 2 we can imagine it to be one bee colony with multiple food resources in it. The main goal of this algorithm is that Employee bees will be helping Unemployed bees to get employed by helping them in finding food resources. So, the Employee bee will start foraging and will do the waggle dance if it finds any food resources, this will help Unemployed bees find the exact location of where the food resource is present and will help get it employed.

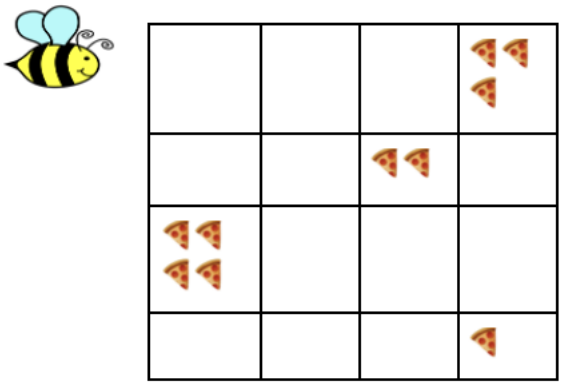


Figure 2: Solution matrix (Artificial Bee Colony with food resources)

Listing 1 is the algorithmic representation for how ABC works like. First we will initialize a random bee population, then the Employee bees will look for Food resources, Onlooker Bees will try to find a place which has maximum food and scout bee will get recruited by consuming the resource which is present at the location sent to it by Employee Bee. This process continues until all the resources are exhausted or is all the Scout bee are employed.

Listing 1: Artificial Bee Colony algorithm

```
1 Initialize Artificial Bee population
2 REPEAT
3       Employee Bee Phase
4       Onlooker Bee Phase
5       Scout Bee Phase
6 UNTIL maximum cycle is reached
```

3.2 Evolutionary Computation

Evolutionary Computation (EC) is a sub-field of bio-inspired computation that uses computational models of natural evolution to solve complex optimization problems. EC algorithms are based on the principles of natural selection and genetic variation and are designed to search for optimal solutions by mimicking the process of natural evolution.

The basic idea behind EC is to create a population of candidate solutions (often represented as individuals or chromosomes) and allow them to evolve through successive generations by applying operators such as selection, crossover, and mutation. These operators mimic the natural processes of selection, reproduction, and genetic variation, respectively.

In each generation, the individuals are evaluated based on a fitness function that measures their quality as solutions to the optimization problem. The fittest individuals are then selected to form the basis for the next generation, while less fit individuals are discarded. The process is repeated for a fixed number of generations or until a satisfactory solution is found.

- Genetic Algorithms (GA)
- Evolution Strategies (ES)
- Genetic Programming (GP)

GA is one of the most widely used EC algorithms. It creates a population of candidate solutions represented as strings of bits or characters and evolves them through selection, crossover, and mutation.

We have worked on implementing Multi agent classification system using Genetic algorithms. We implemented a multi-agent classification system that can tackle large datasets where each agent independently explores a random small portion of the overall dataset, searching for meaningful clusters in proper sub-spaces where they are well-formed. This search is orchestrated by means of a genetic algorithm able to act in a multi-modal fashion, since meaningful clusters might lie in different sub-spaces.

For implementing this classification system we have used Improved Evolutionary Agent Based Clustering Classifier algorithm (E-ABC2) which solves both clustering and classification problems. This algorithm ensures that each cluster contains only features that are similar to each other and avoids outliers which helps in formation of meaningful clusters.

The Evolutionary Agent Based Clustering Classifier (E-ABC2) algorithm combines elements of evolutionary computation, agent-based modeling, and clustering techniques to perform classification tasks.

The E-ABC2 algorithm starts by randomly generating a set of agents that act as individual classifiers. These agents then compete with each other to classify the data points in the input dataset. During this process, the agents adjust their parameters and clustering techniques to improve their classification accuracy.

The E-ABC2 algorithm also includes an evolutionary process where we have used Genetic algorithm to perform selection, crossover and mutation mechanisms on agents and help them produce the next generation. This process continues until the algorithm converges to a set of clusters that produce accurate classifications on the input data.

One of the benefits of the E-ABC2 algorithm is that it can handle complex, high-dimensional datasets that may be challenging for other clustering algorithms.

The reason why we are implementing this application using a multi agent system is because comprehensive research has proved that clustering algorithms when combined with multi agent systems produced some impressive results in terms of scalability, flexibility and robustness.

- Multi-agent systems can be scaled to handle large, complex datasets by distributing the computational load among multiple agents.
- Agents in a multi-agent system can be designed to perform specific tasks and adapt to changes in the environment or the data being analyzed.
- Multi-agent systems can continue to function even if one or more agents fail or are unavailable.

In [9] each agent runs on a different clustering algorithm in order to return the best clustering algorithm for the dataset at hand.In [10] a genetic algorithm has been used where the agents'

genetic code is connection-based: each agent is a clustering result whose genetic code builds a subgraph and, finally, such subgraphs can be interpreted as clusters.

In E-ABC2 each agent runs a very simple clustering procedure on a small sub-sample of the entire dataset. A genetic algorithm orchestrates the evolution of agents in order to return a set of well-formed clusters, thus discovering possible regularities in the data set at hand.

Many clustering algorithms deal with a global metric. Global metrics measure the overall quality of clustering at a global level. They evaluate the clustering results based on the entire dataset, rather than individual data points or clusters. This metric is useful for comparing the performance of different clustering algorithms or parameter settings, and for determining the optimal number of clusters. In the proposed approach we use *"local metric"* to determine the quality of the clusters formed. Local metrics evaluate the clustering results based on individual data points or clusters, rather than the entire dataset. This metric is useful for identifying misclassified or poor clusters, and for identifying subgroups within clusters.

3.3 Ecological Computation

Ecological Computation (EcoC) is a subfield of bio-inspired computation that draws inspiration from ecological systems and their behavior. The main idea behind EcoC is to create computational models that mimic the interactions between species in an ecosystem and use these models to solve optimization problems.

The basic concept behind EcoC is to simulate the interactions between different species in an ecosystem to find optimal solutions to an optimization problem. In these simulations, individuals or species are represented as agents that interact with each other based on their individual traits and environmental factors.

EcoC algorithms often rely on concepts such as competition, cooperation, and adaptation to find optimal solutions. For example, in a competitive ecosystem, different species may compete for resources, and those that are better adapted to their environment will survive and reproduce. Similarly, in a cooperative ecosystem, different species may work together to achieve a common goal. Some examples of EcoC are:

- Bio-geography based optimizer (BBO)
- Invasive Weed Colony Optimizer (IWCO)
- Multi-Species Optimizer (MSO)

Bio-geography-based optimization (BBO) is a nature-inspired algorithm it is based on the principles of bio-geography, which is the study of the distribution of living organisms on earth.

BBO mimics the process of bio-geography by modeling the problem solution as a set of candidate solutions that represent different islands in a geographical map. Each island is associated with a specific solution to the problem and its quality is measured by an objective function. The aim of the algorithm is to find the optimal distribution of the solutions on the islands, such that the overall objective function is minimized.

The BBO algorithm consists of two main stages: *"migration"* and *"mutation"*. In the migration

stage, the algorithm models the process of immigration and emigration of organisms between islands. This process is based on a mathematical model that takes into account the quality of the solutions and the distance between the islands. The *"migration"* process aims to transfer the best solutions from one island to another, in order to improve the overall quality of the solution.

In the *"mutation"* stage, the algorithm models the process of mutation that occurs in living organisms. This process is based on a mathematical model that introduces random changes to the solutions in order to explore new regions of the solution space. The mutation process aims to improve the diversity of the solutions and prevent the algorithm from getting trapped in local optima.

We have worked on implementing the JSSP which is a critical issue in production planning and control. Numerous manufacturing applications in the real world require a set of jobs to be scheduled on a set of machines in order to maximize a particular objective function, such as reducing the total completion time or the makespan.

Job Shop Scheduling issues can be effectively resolved, which can increase customer satisfaction, lower production costs, and significantly increase manufacturing productivity. Additionally, the need to solve these issues in real-time are growing as manufacturing processes is becoming more complex and dynamic.

Furthermore, it is well known that job shop scheduling issues are NP-hard, which means that they are computationally intensive and challenging to solve in an efficient manner. So, creating efficient and effective algorithms to address these issues is a crucial area of research.

Other categories of Bio inspired algorithms such as GA from Evolutionary computation and PSO from Swarm based computation may not work well when the search space is discrete since they are developed for continuous optimization situations.

GA and PSO are prone to get stranded in local optima, especially while working with challenging optimization issues like JSSP. This could result in less than ideal solutions.

The JSSP has been solved effectively using the Bio-geography Based Optimization (BBO)

algorithm, which is built to tackle discrete optimization problems. The biogeography idea that species migration and evolution may be treated as an optimization process is the foundation of the BBO algorithm. The BBO algorithm explores the search space and prevents local optima entrapment by using the principles of immigration and emigration. It has been demonstrated to offer superior outcomes for JSSP than GA and PSO.

4 Applications

This section provides an in-depth description about what are the core differences between Artificial Bee Colony, Genetic Algorithm and Bio-geography based Algorithm. It also describes how we have designed the sequential and agent based versions of the applications.

Table 1 depicts some important differences between each of the algorithms.

Table 1: Difference between ABC, GA, BBO

Algorithm Type	**Artificial Bee Colony (ABC)**	**Genetic Algorithm (GA)**	**Bi-geography Based Optimization (BBO)**
Definition	Swarm-based optimization algorithm inspired by the foraging behavior of honey bees	Optimization algorithm based on the principles of natural selection and genetic operations	Optimization algorithm inspired by the processes of immigration and emigration in biogeography
Inspiration	Inspired by the Foraging behaviour and communication between another honey bee in a hive	Inspired by the principles of genetics, inheritance, and natural selection observed in biological evolution.	Inspired by the immigration and emigration processes of species in biogeography
Main Components	Types of Bees (employed, onlooker, and scout), employed bee phase, onlooker bee phase, fitness evaluation	Population of individuals, fitness evaluation, selection, crossover, mutation	Habitats, migration, immigration, emmigration, mutation, fitness evaluation
Search Strategy	Utilizes employed bees, onlooker bees, and scout bees to explore and exploit the solution space	Implements selection, crossover, and mutation operators to explore and exploit the solution space	Simulates the migration and mutation processes to find optimal solutions
Application Areas	Image Processing, Artificial Intelligence	Clustering and Classification, Machine Learning	Combinatorial optimization problems, Portfolio Optimization

4.1 Image Segmentation using Artificial Bee Colony

ABC for Image Segmentation

Step 1: Convert the input 16 bit color image into grayscale, and then obtain histogram for grayscale image.

Step 2: Initialize random population

- will be initialiting N # of random solutions.
- Will be segmenting the image into D classes(will obtain D intensities of colors by end).
- Each class will contain 3 variables (probability, standard deviation and mean)

After initializing the population , each row would look something like below.

$$I_N = \{P_1^N, \sigma_1^N, \mu_1^N, P_2^N, \sigma_2^N, \mu_2^N, P_3^N, \sigma_3^N, \mu_3^N\} \tag{1}$$

P $\rightarrow Probability$; $\sigma \rightarrow StandardDeviation$; $\mu \rightarrow mean$;

N $\rightarrow numberOfSolutions$; $D \rightarrow numberOfSegments$

Step 3: Implement ABC

REPEAT

Employee Bee 1(a) will find next best solution using the equation listed below

$$v_{j,i} = x_{j,i} + \phi_{j,i} \times (x_{j,i} - x_{j,k}) \tag{2}$$

$$\text{k} \in \{1, 2, ..., N_p\}; j \in \{1, 2, ..., D\}$$

j $\rightarrow$ $D \# of\ attributes\ in\ a\ solution$; $k \rightarrow$ $randomly\ pick\ one\ solution$;

$\phi_{j,i} \rightarrow$ $random\ penalty$; $x_{j,i} \rightarrow$ $value\ of\ jth\ attribute\ in\ ith\ solution$;

$x_{j,k} \rightarrow value\,of\,jth\,attribute\,in\,kth\,solution; \qquad v_{j,i} \rightarrow value\,of\,jth\,attribute\,in\,new\,solution$

After replacing all the values of $v_{j,i}$ in old solution, Employee Bee will form a new solution.

Employee Bee 1(a) will replace the new solution with the original solution only if the fitness value is greater than previous fitness value.

Onlooker Bee 1(b) picks a solution which has a high probability of having a good fitness score.

if Employee Bee 1(a) cannot produce a new solution after x iterations, the solution is abandoned and

Scout Bee 1(c) will generate a new random solution

UNTIL max iterations are reached.

Fitness Score Computation

We need to calculate gaussian probability and calculate the objective function of each solution, using these metrics we will be able to calculate the fitness value of the selected solution.

1. calculate Gaussian probability function

$$p(x) = \Sigma_{i=1}^{D} P_i.p_i(x) = \Sigma_{i=1}^{D}(P_i \div \sqrt{2\pi\sigma_i}) * exp[-(x-\mu)^2 \div 2\sigma_i^2] \tag{3}$$

p(x) $\rightarrow probability\,of\,one\,solution\,out\,of\,N\,solutions$

$P_i \rightarrow$ $summation\,of\,probabilities\,of\,all\,the\,D\,attributes\,of\,a\,solution$

$P_i(x) \rightarrow probability\,of\,a\,class\,in\,ith\,solutuion$

$\sigma_i \rightarrow standard\,deviation\,of\,one\,class\,in\,ith\,solution$

x $\rightarrow number\,of\,pixels\,in\,an\,image$;

$\mu \rightarrow mean\,of\,one\,class\,in\,ith\,solution$

2. Using gaussian probability calculate objective function

$$J = \Sigma_{j=1}^{n}[p(x_j) - h(x_j)]^2 + \omega * |[\Sigma_{i=1}^{D} P_i] - 1| \tag{4}$$

$\mathrm{p}(\mathrm{x}_j) \rightarrow gaussian\, probability\, of\, current\, solution.$

$\mathrm{h}(\mathrm{x}_j) \rightarrow gaussian\, probability\, of\, originalImage\, solution.$

$\omega \rightarrow constant\, penalty$

$\mathrm{J} \rightarrow fitness\, value\, of\, a\, solution$

3. Calculate the fitness value

$$\mathrm{fit}_i = 1/(1 + J_i) \qquad if J_i >= 0$$

$$fit_i = 1 + abs(J_i) \qquad if J_i < 0 \tag{5}$$

After reaching the threshold limit number of iterations the ABC algorithm implementation will be stopped and the best solution will be picked based on the fitness score.

After finding out the best solution we need to find out the pixel intensity to which we want to plot our output image, for which we need to calculate the thresholds of each class.

Step 4: Calculate thresholds (if we divide image into 3 segments we will get 2 thresholds).

$$AT_h^2 + BT_h + C = 0 \tag{6}$$

$A = \sigma_h^2 - \sigma_{h+1}^2$

$B = 2(\mu_h * \sigma_{h+1}^2 - \mu_{h+1}^2 * \sigma_h^2)$

$C = (\mu_{h+1} * \sigma_h)^2 - (\mu_h * \sigma_{h+1})^2 + 2 * (\sigma_h * \sigma_{h+1})^2 * ln[\sigma_{h+1} * P_h \div \sigma_h * P_{h+1}]$

$P_h \rightarrow Probability\, of\, one\, class$

$\sigma_h \rightarrow StandardDeviation\, of\, one\, class$

$\mu_h \rightarrow mean\, of\, one\, class$

Step 5: convert the best solution into a grayscale image.

Image Segmentation with ABC in MASS

The ABC algorithm in MASS uses static agents to run the algorithm as many times as required over a matrix in parallel. The code for the algorithm is divided into 4 Java classes:

1. **Bee :** this class will define the behavior of the Agent Bee which is the static agent construct and defines the methods on the Agent. The Agent will Initialize the population using $init()$ method and will find the next solution using $findNextSolution()$ method.

2. **ImageSegmentationWithABC :** instantiates a multi-dimensional distributed array of the Places construct using *size* argument and initializes it in parallel. For instantiating the Artificial Bee population we use static Agent Bee to create a random population consisting of 20 solutions and then tries to find a solution which is similar to original histogram[8].

Listing 2 depicts the static Agent Bee behavior as described in Bee class.

Listing 2: ABC Agent Behavior in MASS

```
1  import MASS.*;
2  public class Bee extends Agent {
3   Initializations for mean, deviation and random seed
4   public Object callMethod(int functionId, Object argument){
5    switch(functionId){
6     case _init: return init(argument);
7     case _findNextSolution: return findNextSolut(argument);
8      }
9   return null;
10  }
11 }
```

After initialization of Places and Agents we maintain a couple of array objects called *prevIterationSolutions* and *toBePassedNextIteration* . These array objects are maintained on main program to keep track of the solutions and help agents in finding, generating new solutions. Agents find the solution by randomly picking a solution from previous iteration which has max fitness score and generates a new solution. After completing all the initializations MASS will perform a parallel function call to all agents using Places.callAll() i.e., *imageThresholds.callAll()*, Agents.callAll() will perform the required computations and will find the final best solution after completion of ABC. Listing 3 depicts the implementation mentioned above.

Listing 3: Main program for ImageSegmentationWithABC in MASS

```
1  import MASS.*;
2  public class ImageSegmentationWithABC {
3  public static void main(String[] args){
4  GrayScaleIntensityConverter imageConverter = new (...);
5  HistogramHelper histHelper = new HistogramHelper();
6  Places imageThresholds = new Places(...);
7  imageThresholds.callAll( ImageThreshold.init_ );
8  Agents bees = new Agents(...);
9  double[][] prevIterationSolutions = new double[nAgents][];
10 double[][] toBePassedNextIteration = new double[nAgents][];
11 double[][] histCopies = new double[nAgents][];
12 for (int ind = 0; ind < nAgents; ind++){
13   histCopies[ind] = hist;
14 }
15 try {
16  for (int i = 0; i < iteration; i++) {
17  Object[] retSolutions;
18  if (i == 0) {
19   retSolutions = (Object[])bees.callAll(Bee._init, histCopies);
20  }
21  else {
22   retSolutions = (Object[])bees.callAll(Bee._findNextSolution,
23                        toBePassedNextIteration);
24  }
25 }}}}}
```

Image Segmentation with ABC in Repast Simphony In Repast Simphony Places are called as Projections(space, grid) and main program is called Context Builder and it doesn't contain a main method. parallelization in Repast Simphony happens using annotations like *@Scheduled-Method(start= x, interval = y)* and we can easily write methods for Agents .

Listing 4 depicts the overall working of Artificial Bee colony for Image Segmentation in Repast Simphony. we have to import repast specific libraries just like in java and need to add all the required Agent objects to context in lines 8,9 we will be adding Employee Bee Agents and OnlookerBee Agents to our context and in line 10 we will be assigning these objects to the grid projection created in line 7. After returning the context is when the program exection starts the scheduled method, which means all the Agents present in the grid will become active and start implementing the ABC algorithm. After completing X number of iterations we will stop the execution and will draw the final image using the Best Solution obtained.

Listing 4: Context Builder for ImageSegmentationWithABC in Repast Simphony

```
1  import repast.simphony.context.Context;
2  import all the repast specific libraries
3  public class ABC implements ContextBuilder<Object> {
4    Initializations for N,D and PIXELS
5    public Context build(Context<Object> context) {
6      Read input image and generate histogram
7      Create Projections(ContinuousSpace and GridFactory)
8      Initialize Employee Bee Agent Objects and add to context
9      Initialize Onlooker Bee Agent Objects and add to context
10     Assign Agents to Grid Projecttion
11     return context;
12     } }
```

Agent Bee Behavior

Listing 5 depicts the typical behavior of a Bee, where Agent Bee will be initialized during the context formation and *@ScheduledMethod* will find the *nextBestSolution* depending on the type of the Bee. if the Agent is of type EmployeeBee then a random solution is selected from the entire search space, the newly selected random solution will help the Agent in finding the best solution among the newly picked random solution and old solution using Fitness Score. Similarly if the Agent is of type Onlooker Bee then a random solution is selected based on fitness scores and it doesnt mean that solution with highest fitness score will be selected all the times, but significantly good solution will be selected from the entire search space.

Listing 5: ABC Agent Behavior in Repast Simphony

```
1  import all the repast specific libraries
2  public class Bee {
3   Initializations for FitnessScoreHelper, space and grid
4   Bee(FitnessScoreHelper fitnessScoreHelper,
5   ContinuousSpace<Object> space, Grid<Object> grid)
6   {
7    this.sol = Initialization.generateNewStart();
8    this.fitnessScore = fitnessScoreHelper.getFitness
9                                          -Value(this.sol);
10  }
11  @ScheduledMethod(start = 1, interval = 1)
12  public void setNextBestSolution(){
13  Find all the neighbouring Agents in the grid
14  select a random solution
15  super.setNextBestSolution(solutions[selectedSolution],
16                                fitnessScoreHelper);
17  }
18 }
```

4.2 Building Agent Based Classification System using Genetic Algorithm

Improved EABC2 for Classification System

the algorithm is divided into 3 phases starting from defining the *EABC2 Agent Behavior*, *Evolutive orchestration and Model synthesis* followed by *Testing phase*.

EABC2 Agent Behavior:

the key role of agent here is to construct well formed clusters. Each agent uses a subset of patterns R, randomly sampled from the training set, to build the decision regions of the classification model M. The clustering algorithm used by each agent is the Basic Sequential Algorithmic Scheme (BSAS), which assigns a pattern to an existing cluster if the pattern-to-cluster distance is below a given threshold θ. If the pattern cannot be assigned, it becomes a centroid for a new cluster. However, BSAS may return a large number of small clusters, especially for low θ values. To address this, a maximum number of allowed clusters can be defined and new clusters can be spawned if the number of available clusters is below the allowed number.
In E-ABC2, a Reinforcement Learning-based BSAS (RL-BSAS) is used to mitigate the sensitivity of BSAS to pattern order and outliers. In RL-BSAS, an energy value is assigned to each cluster. When a cluster receives a new pattern, its energy is increased by a value $\alpha \in [0,1]$, while the energies of all other clusters are diminished by a value $\beta \in [0,1]$. Relevant clusters will survive with high energies, while badly-formed clusters will eventually vanish when their energies approach zero.

Each E-ABC2 agent depends on a set of parameters:

- the maximum threshold θ for the BSAS procedure
- a binary mask $\mathbf{W} \in [0,1]$ is in charge of selecting only relevant features, defining the specific local metric used by the agent in searching for well-formed clusters. ex: [1,0,0,1]
- the ratio parameter r = $\alpha/\beta \in (0,1)$ involved in the RL-BSAS.

Each agent uses the parameters described above to cluster the input data subset R. Initially, it

selects features from the patterns in R that belong to a value of 1 in the binary weight vector W. Then, it applies the RL-BSAS algorithm using the threshold value θ, along with penalty and reward factors α = r·β. For simplicity, the value of α is set to 1, allowing the value of β to be easily calculated as β = α/r. The agent's primary objective is to generate a partition P, which is obtained by running RL-BSAS.

Listing 6: E-ABC2 Agent behavior

```
P - Partition of clusters {C1, . . . ,Cp }
d - distance betwen patterns
R - agent data shard
procedure AGENT_EXECUTION
    P = RL-BSAS (R)
    repeat
    for each pair Ci, Cj in P do
        Evaluate D = d(Ci,Cj)
    until all patterns in R are assigned to a cluster
```

Evolutive Orchestration and Model Synthesis:

Initially each agent will be assigned a random genetic code which contains the set of parameters mentioned above. The genetic code would look like this : [$theta$, W, r].

Each agent will evolve over the time using Genetic algorithm functionalities like Selection, Crossover and Mutation.

Selection: we have collected all the genetic codes from previous generation and sorted the agents based on their fitness scores in non-increasing order. Then we randomly select one strongest parent from the first half and another parent from second half of the agents population that we have. This will help us produce half the amount of population that we previously had. After this, we will randomly select 2 parents from the stronger population that is the first half and produce the off-springs until the desired population is reached. By following this procedure we will be able to produce good quality off-springs.

genetic code structure {[binary mask] , threshold, energy }

Ex: Lets assume the selected parents are : {[1,0,1,1], 0.7 ,9 } and {[0,0,1,1], 0.2 ,4 }

Crossover: In crossover between 2 agents we have randomly selected one parameter each from the 2 input genetic codes that we have and performed mutation on the result. Ex: parents are : {[1,0,1,1], 0.7 ,9 } and {[0,0,1,1], 0.2 ,4 }

Offspring will be : {[1,0,1,1], 0.7 , 4 }

Mutation: for the parameters $/theta$ and r, we have randomly changed the threshold 20% higher or lower. for the binary mask we randomly selected one index and inverted it.

Ex: if the Offspring is : {[1,0,1,1], 0.7, 4 }

After Mutation : {[1,0,0,1], 0.56, 4.8 }

After performing all the above operations each agent exploits the parameters written in its genetic code in order to return a partition P of the random data shard R. As P = C1, . . . ,Cp is returned, each cluster C in the partition is evaluated using quality index $f_{cc}(C)$ that considers both its compactness $f_{co}(C)$ and cardinality $f_{ca}(C)$ of the clusters.

$$f_{cc}(C) = \lambda.f_{co}(C) + (1 - \lambda).f_{ca}(C) \tag{7}$$

$$\mathrm{f}_{co}(C) = ((1 - (\Sigma_{x \in C} d(x, \mu)/|C|)) - co_{min})/co_{max} - co_{min}$$

$$\mathrm{f}_{ca}(C) = |C| - ca_{min}/ca_{max} - ca_{min}$$

$$\lambda \rightarrow user\, defined\, tradeoff\, parameter$$

$$\mathrm{C} \rightarrow number\, of\, instances\, in\, a\, cluster$$

$$\mathrm{d(x,}\mu) \rightarrow sum\, of\, distance\, between\, all\, instances\, in\, a\, cluster$$

co_{min}, co_{max}, ca_{min} and ca_{max} are the minimum and maximum compactness and cardinality values observed during the evolution, μ denotes the centroid of the cluster and $\lambda \in$ [0,1] is a user-defined trade-off parameter. In order to retain only good quality clusters, for each agent,

only the best cluster C^* is retained, namely the cluster in P that maximizes 7

After each agent finds out the best cluster , now we have to evaluate whether the best cluster collected by each agent can be added to our final model or not. This involves finding out the ratio of validation data that share the same label as C and fall within C, to the total number of validation set patterns falling in C.

If $Acc(C)^*$ is greater than both a user-defined threshold Acc_{min} and an adaptive parameter Acc_{gl}, then the cluster C^* is considered to be of good quality and is added to the classification model, denoted as M. M is incrementally built generation-by-generation, and serves as the core of the classification model. Once all agents have offloaded their good clusters in M, these clusters can be considered as building blocks for the classification model.

Listing 7: E-ABC2 Agent Fitness Evaluation and Model Design

```
1  P_i - Partition of clusters C_1,...,C_p of the ith agent
2  R_i - data shard of ith agent
3  <C> - cluster compactness
4  |C| - cluster cardinality
5  procedure AGENTS_EVALUATION
6      Compute Acc_gl for the model M on S_vl
7      for each agent a_i in P do
8          P_i = agent_Execution(R_i)
9          for each C in P_i do
10             f_cc(C) = λ · h_Ci + (1 − λ) · |C|
11         best Cluster{C} = cluster with max f_cc
12         Compute cluster accuracy Acc(C*) on S_vl
13         f = Acc_gl · Acc(C*) + (1 − Acc_gl) · f_cc(C*)
```

Training Phase: Here we start by initializing an empty model M, and initializing a population P_i with randomly generated genetic codes. The algorithm then proceeds through a number of generations to evolve the population towards an optimal solution.

In each generation, the algorithm first draws N random shards from the training dataset S_{tr}. Then, the population P_i undergoes an evaluation step using the function agentsEvaluation to generate a set of clusters, each with a corresponding fitness value f. Then we check the accuracy of the current model M on the validation dataset S_{vl}, denoted as Acc_{gl}. For each agent in the population, the algorithm checks if the accuracy of the best cluster C^* generated by that agent is greater than a user-defined threshold Acc_{min} and greater than the adaptive parameter Acc_{gl}. If so, then we add the cluster C^* to the model M.

Overall, the E-ABC2 algorithm provides a robust and efficient method for training classification models using a combination of optimization and adaptive parameter tuning.

Listing 8 defines the working of how classification model is built.

Listing 8: E-ABC2 Training Phase

```
N - Number of agents
P_i - population of i^th generation
R_j - data shard for j^th agent
procedure TRAINAGENTS
    Set M = ∅
    Initialize individuals in P_0 with random genetic code
    for i = 0 to NG do
        Draw random shards {R_1, ..., R_N} from S_tr
        C*, f = agentsEvaluation(P_i, {R_1, ..., R_N}, S_vl)
        Compute accuracy Acc_gl of model M on S_vl
        for each agent a_j do
            if Acc(C*) > Acc_min && Acc(C*) > Acc_gl then
                Add C* to model M
        P_{i+1} = Evolve(P_i)
```

Multi Agent Classification System using E-ABC2 in MASS

The E-ABC2 algorithm in MASS uses static agents to run the algorithm until maximum accuracy is achieved on a matrix in parallel. The code for the algorithm is divided into 4 major Java classes:

1. **EABC2Main:** instantiates a multi-dimensional distributed array of the Places construct using *size* argument and initializes it in parallel. For instantiating the Agent population we use *MLClassificationAgent* to create N agents with random genetic codes and evolve until all the patterns in training set have been clustered.

2. **MLClassificationAgent:** it calcuates the *Euclidean Distance* between the pattern from training set and cluster centroid, and handles energy management i.e.. assigning penalties to the clusters.

3. **Cluster Evaluation:** it finds the best cluster from all the cluster an agent has formed by calculating the cluster quality. After finding the bestCluster from an agent , it will find the cluster accuracy by mapping it against validation dataset.

4. **Evolutive Orchestration:** takes care of performing selection, crossover and mutation operations.

Listing 9 depicts the Implementation of Multi Agent Classification system in MASS. Lines 7,9 define the initializations of places and Agents objects, followed by the generation of random genetic codes in Line 14,15. In Line 16 we are assigning each agent with its genetic code and training data to form clusters. Line 17,18 depict how we have parallelized cluster formation for each agent using *callAll()* method. After each agent has formed its own set of clusters, we will pick one best cluster from each agent which has highest cluster accuracy which are defined by cluster compactness and cardinality which id shown in Line 20. After collecting one best cluster from all agents we will validate the clusters using validating data which is shown in Line 21. In line 23 we will be calculating accuracy of the model that we are building using validation data, and in Line 25 we are adding the cluster to final model.

Listing 9: E-ABC2 Main program to implement a Classification System in MASS

```
1  import MASS.*;
2  public class EABC2Main {
3  public static void main(String[] args){
4   Initialize iterations, MAX_CLUSTERS, globalMinAccuracy
5   GeneticCodeGenerator generator = new GeneticCodeGenerator();
6   MLClassificationAgent agent = new MLClassificationAgent();
7   MLModel mlModel = new MLModel();
8   Places places = new Places( ....);
9   places.callAll( MASSPlace.init_ );
10  Agents classificationAgents = new Agents(...);
11  try {
12   double[] fitness = new double[nAgents]
13   while (iteration < iterations) {
14    List<GeneticCode> geneticCodes =
15    generator.generateGeneticCodes(nAgents, fitness);
16    for each agent assign its genetic code and train data
17     Object[] responses = (Object[]) classificationAgents.
18     callAll(MLClassificationAgent._init, arguments);
19     for (int i = 0; i < responses.length; i++) {
20      Cluster bestCluster = (Cluster) responses[i];
21      double accuracy = bestCluster.getAccuracy();
22      double globalModelAccuracy = mlModel.getModelAccuracy(
23                          geneticCodes.get(i).getValidation());
24       if (accuracy>globalMinAccuracy &&
25           accuracy>globalModelAccuracy) {
26         mlModel.addToModel(bestCluster); }}
27     iteration++;}}}
28 }
```

Multi Agent Classification System using E-ABC2 in Repast Simphony

In Repast Simphony Places are called as Projections(space, grid) and main program is called Context Builder, parallelization in Repast Simphony happens using annotations like @Scheduled-Method(start= x, interval = y). Listing 10 depicts the overall working of building Classification System using E-ABC2 in Repast Simphony. we have to add all the required Agent objects to context like in line 7 we will be adding Agents,MlModel and Generate random population(Generator) and in line 8 we will be assigning these objects to the grid projection created in line 6. After returning the context is when the program execution starts the scheduled method, which means all objects in context will start implementing E-ABC2 algorithm until it achieves maximum accuracy.

Listing 10: Context Builder for Classification system using EABC2 in Repast Simphony

```
1  import repast.simphony.context.Context;
2  import all the repast specific libraries
3  public class EABC2 implements ContextBuilder<Object> {
4   public Context build(Context<Object> context) {
5    read input dataset and perform sampling
6    Create Projections(ContinuousSpace and GridFactory)
7    Initialize Agent, mlModel Objects and add to context
8    Assign Agents to Grid Projecttion
9    return context;
10 } }
```

Listing 11 depicts the behavior of a Agent, where Agent will be initialized during the context formation and @ScheduledMethod will start the clustering process by first subsampling the data into Train,Test and Validation datasets. All the agent objects will start forming clusters using training data and best cluster will be selected from each agent for which global model accuracy will be calculated and added to the *mlmodel* accordingly.

Listing 11: E-ABC2 Agent Behavior in Repast Simphony

```
1  import all the repast specific libraries
2  public class Agent {
3    Initializations for globalMinAccuracy, space and grid
4    public Agent(int id, GeneticCodeGenerator generator,
5    MLModel mlModel, Dataset dataset)
6    {
7      setup the instance variables : id, generator,
8      clusterEvaluation , mlModel, dataset.
9    }
10   @ScheduledMethod(start = 1, interval = 1)
11   public void init(){
12     Assign Train,Test and Validation data to Agent
13     List<Cluster> clusters = execution(trainData,
14     generator.getGeneticCode(agent_id));
15     Cluster bestCluster = clusterEvaluation.
16     findBestAgentCluster(clusters);
17     double accuracy = clusterEvaluation.
18     bestClusterAccuracy(bestCluster, validationData);
19     double globalModelAccuracy = mlModel.getModelAccuracy(
20                                  validationData);
21     if (accuracy > globalMinAccuracy &&
22         accuracy > globalModelAccuracy){
23         mlModel.addToModel(bestCluster);}}
24   }
25 }
```

4.3 Job Shop Scheduling Problem using Bio-geography Based Optimization

Job Shop Scheduling Problem (JSSP) is where each job consists of a series of tasks that must be completed in a particular order and on a particular machine. For instance, the task can involve producing a single consumer good eg: a vehicle. The challenge is to plan the tasks on the machines so that the total time required to complete everything on the schedule is kept to a minimum and it follows the below rules

1. Before beginning a work for a job, the prior task must be finished for that job.
2. Only one task can be completed by a machine at once.
3. Once a task has begun, it must continue until finished.

we have worked on implementing a general JSSP, assuming that we will be having equal number of operations for all the jobs and that the number of operations will be equal to the number of machines. To understand more about JSSP refer to [11].

op-machine :

	op #		
job	2	1	3
	2	3	1
	3	2	1

jobCost-machine :

	machine #		
job	10	6	3
	5	7	4
	9	13	8

Example of a habitat : { 1, 2, 1, 3, 2, 1, 3, 2, 3 } - defines the sequence of operations

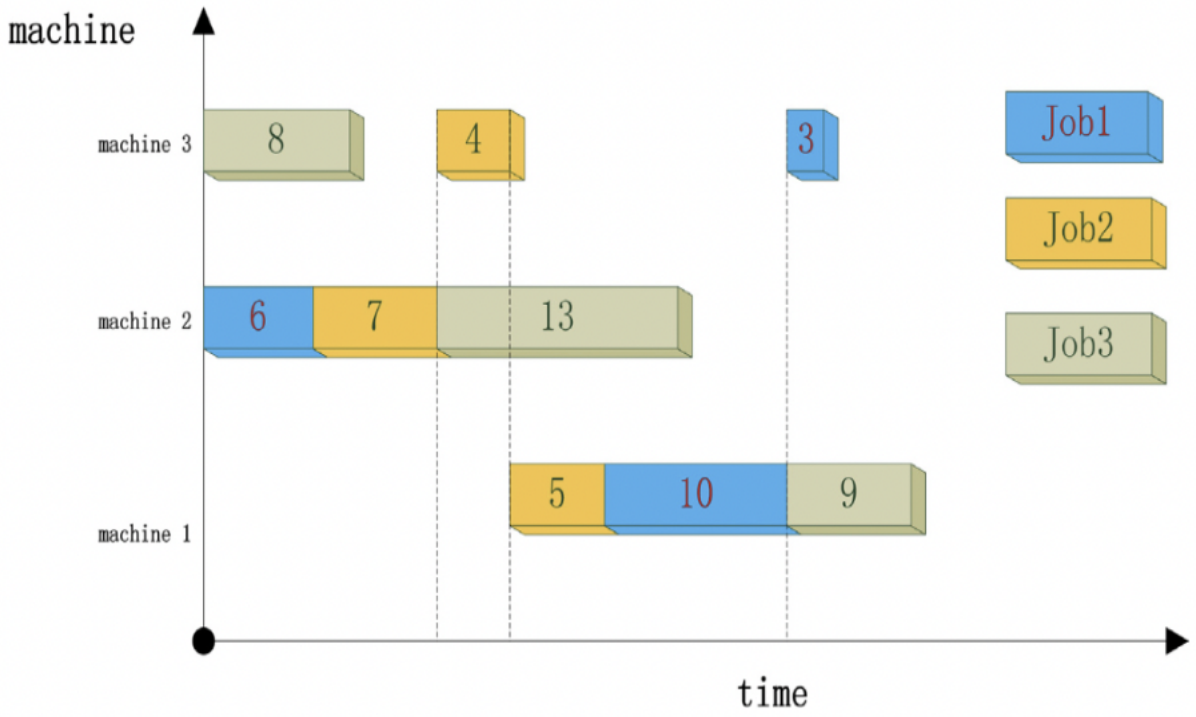


Figure 3: Example of 3*3 JSSP

The sequence of operations executed on machines would look like following:

1. Job 1 Operation 1 will run on machine 2, because there are no jobs running on machine2.
2. Job 2 Operation 1 will run on machine 2 only after machine 2 becomes free.
3. Job 1 Operation 2 will run on machine 1, only after the first operation of Job 1 has been executed and there are no jobs running on machine 1.
4. Job 3 Operation 1 will run on machine 3, because there are no jobs running on machine 3.
5. Job 2 Operation 2 will run on machine 3 only after the first operation of Job 2 has been executed and there are no jobs running on machine 3.

The process continues until all the Operations of all the Jobs have been scheduled. The total makespan of the above habitat is 41s.

BBO for JSSP

BBO mimics the geographical distribution of species, how new species emerge, and how they become extinct. Each group of these distributions are called habitats and each habitat can be considered as a potential solution. If a habitat is suitable for biological survival, it is determined using a quantitative performance index called the Habitat Suitability Index (HSI).

The candidate solutions in the BBO are presented using a collection of habitats. Migration and mutation are the two main mechanisms used by the BBO. In various population-based optimization techniques, fitness is the criterion for determining whether a solution is good. HSI is the equivalent of fitness score in BBO [12].

Step 1 : Initialize Habitats / Encoding

The JSSP's habitat encoding involves representing it as a sequence of **n** job assignments to **m** machines, i.e., {j1, j2, j3, j2, j1, ... , jn*m}.

Due to the large number of jobs and machines involved, the total number of possible habitats is (jobs * machines)! , making this problem NP-hard. To tackle this issue, we can restrict the number of habitats to be considered in the candidate solution set and employ BBO to efficiently identify the best habitat among them.

Ex: lets assume we have jobs = 2 and machines = 2, and our habitat count is 3.

habitats : { 1, 2, 2, 1 } , { 1, 2, 1, 2 } , { 1, 1, 2, 2 }

Step 2 : Compute Habitat Suitability Index

After generating required number of habitats, now we have to calculate the suitability index or fitness value for each each habitat.

1. Maintain a Hashmap to keep track of the operations on a job.
2. Find out the job number (J) and its operation number (i).
3. Find out which machine(M_i) the current job should run on.
4. Now, Find the end time of current job previous operation (J_{i-1}), and the end time of the job running on the current machine (M_i).
5. update the currJobStartTime as max(end time of M_i , end time of (J_{i-1}) .

6. maintain start and end times of current operation of a job (J_i) , and the operation running on machine (M_i).

Step 3: Avoid Stagnation

Maintain stagnation count of the HSI's to avoid the stagnation of the solution. If a habitats HSI hasn't changed in past x iterations then we change the features of the solution to help it escape by getting trapped in local optima.

Step 4: Calculate Immigration Eligibilities of habitats

1. Normalize HSI's for all the habitats:

$$normalizedHSI[i] = ((computedHSIs[i] - minHSI)/(maxHSI - minHSI)) \quad (8)$$

2. probabilistically assign the immigration eligibility to each of the habitats, habitats will lower normalized HSI's will have high probablity to immigrate.

   - if currHSI $<$ globalMinHSI, then we will allow the habitat to immigrate.

   Ex: Lets assume we have a habitat of size 3 then immigration = { false, false, true }.

Step 5: Calculate Emigration Eligibilities of habitats

1. Normalize HSI's for all the habitats:

$$normalizedHSI[i] = 1 - ((computedHSIs[i] - minHSI)/(maxHSI - minHSI)) \quad (9)$$

2. Probabilistically assign the emigration eligibility to each of the habitats. Here we inverted the normalization because Habitat with lower HSI will have higher probability to emigrate.

   - if currHSI $>$ globalMaxHSI, then we will allow the habitat to emigrate.

   Ex: emmigration = { true, false, false }.

Step 6: Perform migration

Now we have to change the habitat with immigration eligibility to look like emigrating habitat

1. iteratively pick habitat with immigration eligibility.
2. pick a random habitat which has emigration eligibility.
3. Assuming we have M number of machines, randomly pick M indexes from emigrating habitat.
4. perform cyclic rotation of values in the habitat. Ex: element present in 1st index will be shifted to 2nd index from the chosen M indexes, element present in 2nd index will be shifted to 3rd index and the process continues.

Ex: habitats : { 1, 2, 2, 1 } , { 1, 2, 1, 2 } , { 1, 1, 2, 2 }, immigration = { false, false, true } and emmigration = { true, false, false }

Lets assume habitat 1 has highb HSI which is why it doesnt immigrate and has probabilistically gained emmigration eligibility. If we observe immigration[2] then we can see that it is eligible to migrate , so lets randomly pick emmigration[0] and perform migration.

{ 1, 2, 2, 1 } is the habitat present at habitat[0].

Now lets pick machines number (2) random indexes in habitat[0] and perform cyclic rotation,Indexes picked are 0 , 2.

After performing migration new habitat will be { 2, 2, 1, 1 }.

Step 7: Recalculate HSI's for newly formed habitats

After performing migration, some of the habitats which previously had lower HSI's were changed. So we have to recalculate HSIs of the new habitats and check immigration eligibilities.

Step 8: Perform mutation

To perform mutation:

1. select a habitat to mutate.

2. randomly pick a chunk of habitat and shuffle the order of operations.

   Ex: Suppose the habitat is { 2, 2, 1, 1 }.

   Lets pick indexes 1,2 as our chunk to rotate.

   After rotating the chunk, this is how the new habitat will look like : { 2, 1, 2, 1 }.

stop when mutation_rate amount of habitats have been mutated.

Repeat the entire process for Y number of iterations to find the shortest makespan formed by a habitat.

Job Shop Scheduling Problem using BBO in MASS

The BBO algorithm in MASS uses static agents to run the algorithm for a certain number of iterations and finds out the shortest makespan of a habitat. The code for the algorithm is majorly divided into 4 Java classes:

1. **JobShopSchedulingProblem:** instantiates a 2D distributed array of the Places construct using size argument and initializes it in parallel. Generates a set of candidate solutions(habitats). *hsiComputationAgents* will be the agent objects that will calculate the HSI of habitats.

2. **HSIComputationAgent:** maintains two methods *_init* and *_computeHSI*. These methods will take care of HSI computation using *HabitatSuitabilityIndex* class.

3. **HabitatUtils:** is an important class that contains all the major computation methods for [*generateRandomHabitat* , *generateNewHabitatsForEliteSolutions*, *getImmigrationEligibility*, *getEmigrationEligibility*, *getMinMax*, *performMigration*, *mutate*.]

Listing 12 depicts the Implementation of Job Shop Scheduling Problem using MASS. Line 5,6 depict that we will be initializing our places matrix and Line 7 depicts the initialization of Agents object. Line 11 depicts that in the 1st iteration we will be randomly initializing the habitats using *generateRandomHabitat()* and calculate the HSI for each of them. In Line 17 we are essentially

dividing the habitats into equal sized groups so that each Agent will get equal load to work on. After diving the habitats among agents Line 20 shows that we are performing parallelization by using *callAll()* method which will distribute the Agents onto multiple remote nodes and all the nodes will start computing the HSI for each of the habitats. Depending on the minimum and maximum HSI's we will be calculating the immigration eligibility which is shown in Line 23 using *getImmigrationEligibility()*, and in Line 24 we will calculate emmigration eligibility using *getEmigrationEligibility()*. By considering the immigration and emmigration elibilities we will implement *performMigration()* as shown in Line 25. After performing migration some habitats will change so we will have to recompute the HSI's as shown in Line 26 and then perform mutation.

Listing 12: BBO main program to depict Job Shop Scheduling Problem in MASS

```
1  import MASS.*;
2  public class  JobShopSchedulingProblem{
3  public static void main(String[] args){
4   Initializations for jobs,machines,habitatCount
5   Places places = new Places( ....);
6   places.callAll( MASSPlace.init_ );
7   Agents hsiComputationAgents = new Agents(...);
8   Generate habitatCount # of candidate solutions
9   try {
10   for(int iter = 0; iter < iterations; iter++){
11    if (iter == 0) {
12     hsiComputationAgents.callAll(HSIComputationAgent._init ,
13                                                          hsi);
14     continue;
15    }
16    Object[] habitatGroups = new Object[nAgents];
17    for (int i = 0; i < nAgents; i++) {
18     int[][]group =Arrays.copyOfRange(currGenHabitats ,
```

```
19                          i * groupSize,((i + 1) * groupSize));
20      habitatGroups[i] = group; }
21      Object[] responses = (Object[])hsiComputationAgents.
22      callAll(HSIComputationAgent._computeHSI, habitatGroups);
23      int[] minMax = habitatUtils.getMinMax(computedHSIs);
24      boolean[]immigration = habitatUtils.getImmigration
25                                          -Eligibility(...);
26      habitatUtils.getEmmigrationEligibility(...);
27      performMigration(...);
28      Recompute HSIs
29      habitatUtils.mutate(...); }}
30      finally {
31        MASS.finish(); }
32  }}
```

Job Shop Scheduling Problem using BBO in Repast Simphony

In Repast Simphony parallelization can be implemented using annotations like *@ScheduleMethod (start = x, end = y)*. Listing 13 describes the flow of how JSSP can be executed using BBO in Repast Simphony. Line 4 depicts that we have to create context using spaceFactory and gridFactory. Line 8 depicts that HSIComputationAgent will create habitats and add the agent objects to the context.

Listing 13: Context Builder for JSSP using BBO in Repast Simphony

```
1  import all the repast specific libraries
2  public class JSSPBuilder impelments ContextBuilder<>{
3    Initialize # of jobs, machines, habitat_count
4    Create context using ContinuousSpaceFactory and GridFactory
5    Initialize hsi , habitatUtils, habitatGenerator and nAgents
6    context.add(habitatGenerator);
7    for (int i = 0; i < nAgents; i++) {
```

```
8       context.add(new HSIComputationAgent(habitatGenerator
9                                                   , hsi, i));
10      }
11    return context; }
12  }
```

Agent parallelization in Repast Simphony

Listing 14 defines the typical behaviour of an agent in BBO. On the execution of Line 10, the initialization of Agents and random habitats will take place, followed by HSI computation by each agent as depicted in Line 2. After the HSI computation we will check if each solution has reached stagnation and then perform migration which is depicted in Line 16 and 17. After migration we will recalculate HSIs and then perform mutation which is depicted in Line 24.

Listing 14: BBO Agent Behaviour for JSSP in Repast Simphony

```
1  public class HSIComputationAgent {
2   @ScheduledMethod(start = 2, interval = 2)
3   public void computeHSI() {
4    int[][] habitats = habitatGenerator.getAgent
5                                -Partition(agentId);
6    habitatGenerator.setAgentHabitats(
7    habitatSuitabilityIndex.computeHSIs(habitats), agentId);
8    }
9  }
10 public class HabitatGenerator {
11  @ScheduledMethod(start = 1)
12  public void init() {
13     generate random habitats
14     int groupSize = habitatCount/nAgents;
15     Each agent will receive groupSize # of habitats
16  }
```

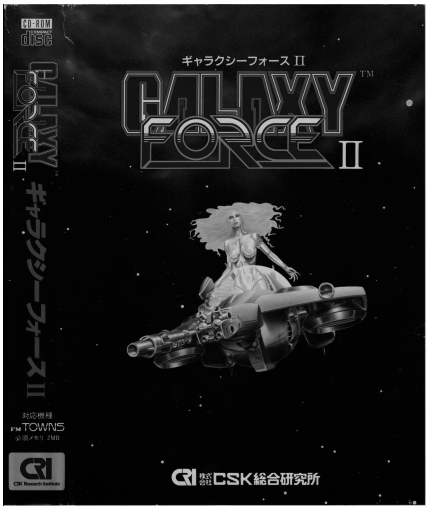
```
17  @ScheduledMethod(start = 3, interval = 4)
18  public void checkStagation(){
19   if (max_stagnated_count == AllowedStagnation){
20     habitatUtils.generateNewHabitatsFor
21     -EliteSolutions(computedHSIs,currGenHabitats);  }
22    performMigration();
23  }
24  @ScheduledMethod(start = 5, interval = 4)
25  public void performMutation(){
26     habitatUtils.mutate(currGenHabitats, immigration);
27  }
28  @ScheduledMethod(start = 4002)
29  public void finalResult() {
30   System.out.println("Best HSI: " + bestHSI); }
31 }
```

5 Performance Measurements

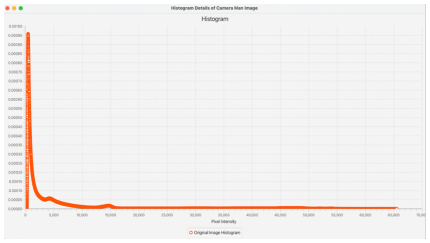
This section presents the results obtained by algorithms implemented in sequential, MASS and Repast Simphony paradigms. It also provides a deeper look into how these paradigms have performed in sequential and parallel settings.

5.1 Artificial Bee Colony Performance

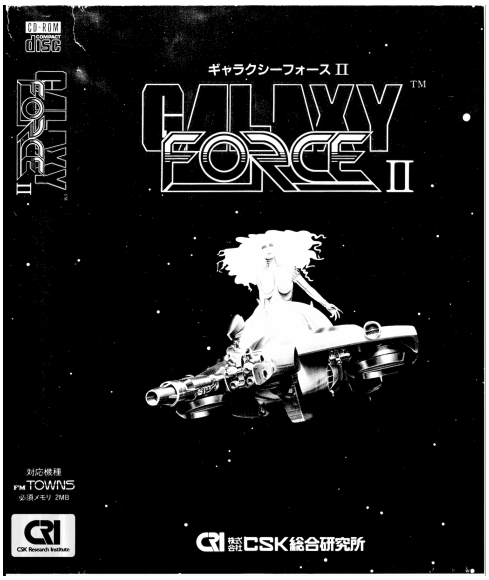
We have considered 16 bit images for measuring the performance .Fig 4(a) , 4(b) depict Input grayscale image and histogram. Fig 4(c) , 4(d) are segmented outputs for 1k and 10k iterations:



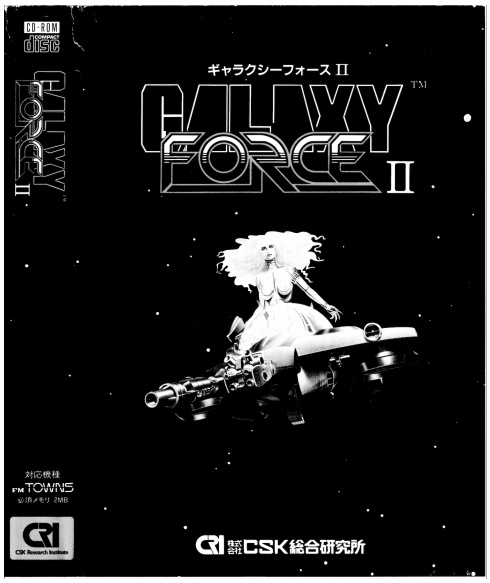
((**a**)) input 16 bit grayscale image



((**b**)) histogram for input image



((**c**)) MASS output for 1k iterations



((**d**)) MASS output for 10k iterations

Figure 4: Results produced by ABC algorithm for Image Segmentation

For this experiment we have considered a Japanese gaming image of size 567.7MB, we can see the difference between the outputs produced for 1K and 10K iterations.

When an image is segmented, it is divided into multiple regions or segments, where each segment corresponds to a distinct object or part of the image. The boundaries between these segments are the edges or transitions where the properties of the pixels change abruptly from one segment to another. If we observe 4(c) we can see that the object of woman on a spaceship is an abrupt transformation when we compare it with its background. These boundaries were characterized by a significant change in pixel intensities.

Now if we observe 4(d) we can say that it has been segmented more clearly because if we observe the woman on spaceship object, previously for 1k iterations small details like mouth and the frills on woman's gown were not well detected, but if we increased the number of iterations many small objects and correct pixel values were identified.

The presence of distinct visual differences can serve as clear and reliable indicators that an image has been effectively segmented into multiple segments. By visually identifying these differences, we can confidently conclude that the segmentation process has successfully partitioned the image into meaningful and distinct clusters, which can be further analyzed or processed as needed.

From Table 2 we can see that by parallelizing the ABC algorithm using MASS, there was significant amount of reduction in time for MASS implementation when compared to Serial Implementation. We could achieve 3.5x speed using single remote cluster on 20 random solutions. But, Repast Simphony was much slower than the other 2 implementations. It is because when every time an agent needs to pick a random solution the agent needs to look for all its neighbors and select one random neighbor. This process happens for all the agents every single time and we are storing all the objects in context which might slow down the program.

Table 2: Time taken to execute 20 solutions with 20 Agents

Time taken	**Serial Implementation**	**MASS Implementation**	**Repast Simphony Implementation**
for 1k iterations	135s	35s	227s
for 10k iterations	1297s	352s	2233s

CPU scalability in MASS for ABC using callAll() on Agents

From Fig 5 we can see that there was a small increase in execution time between 1 computing node and 2 computing nodes, but the execution time linearly kept increasing with the increase in number of nodes and dropped when the # of computing nodes was 24. So this increase might have caused due to performing *callAll()* on agents every single time which is causing a synchronization barrier and is resulting in time delays.

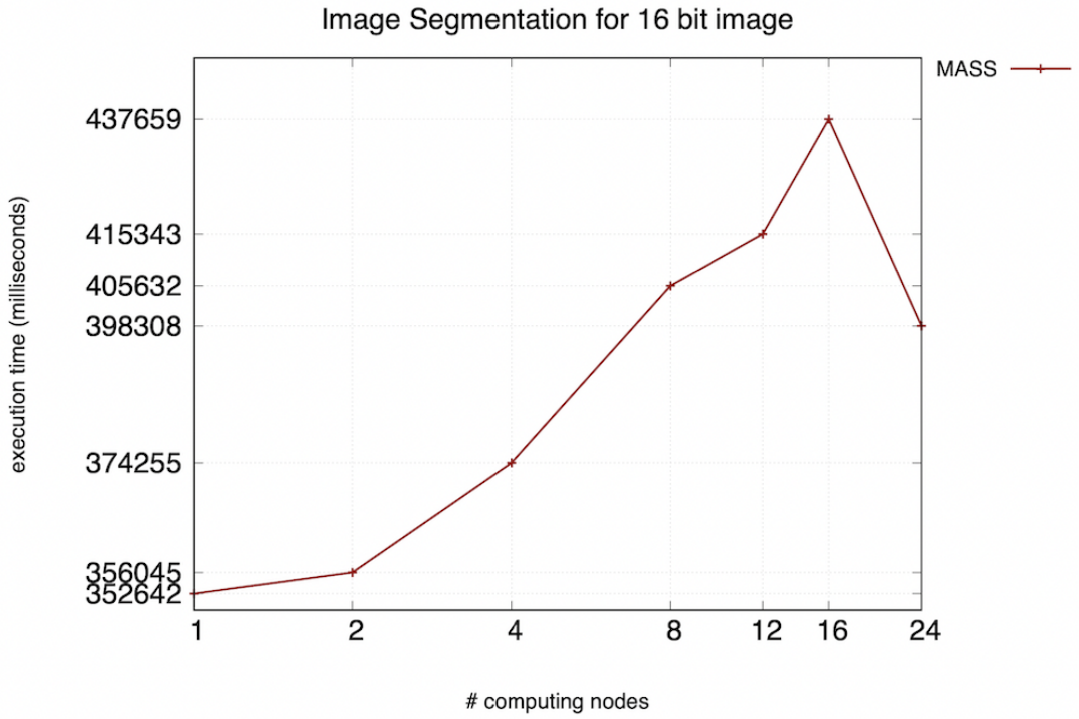


Figure 5: Scalability of ABC using 20 places and 20 Agents over 24 remote nodes

5.2 Genetic Algorithm Performance

We have tested the application with 2 datasets named *"iris.data"* and *"DetectingPhishingWebsites.data"*. Iris is a small dataset with 150 rows and 4 attributes whereas DetectingPhishingWebsites is a large dataset with 11.5k rows and 30 attributes.

The application worked well for the *Iris* dataset in Sequential, MASS and Repast Simphony. The execution accuracy and speed were much better and faster in Repast Simphony, but when large dataset was used both sequential and Repast Simphony took a lot of time for execution where as MASS was much faster in that case.

Tables 3 and 4 show the execution time taken to run the application for 1000 iterations and accuracy achieved by each mode of implementation.

Table 3: Time taken to execute iris dataset with 6 Agents

Time taken	**Serial Implementation**	**MASS Implementation**	**Repast Simphony Implementation**
for 1k iterations	62.24s	10s	5s
Accuracy achieved	75%	86%	86%

Table 4: Time taken to execute DetectingPhishingWebsites dataset with 6 Agents

Time taken	**Serial Implementation**	**MASS Implementation**	**Repast Simphony Implementation**
for 1k iterations	>5hrs	715.068s	>5hrs
Accuracy achieved	83%	90%	87%

CPU Scalability in MASS for GA using *callAll()* on Agents :

The CPU scalability for MASS agents using *callAll()* was a little variable and performed the best when it was implemented on a single node because of the functionalities implemented on Agents. This variability have been caused because of network delay when communicating with agents on multiple machines.

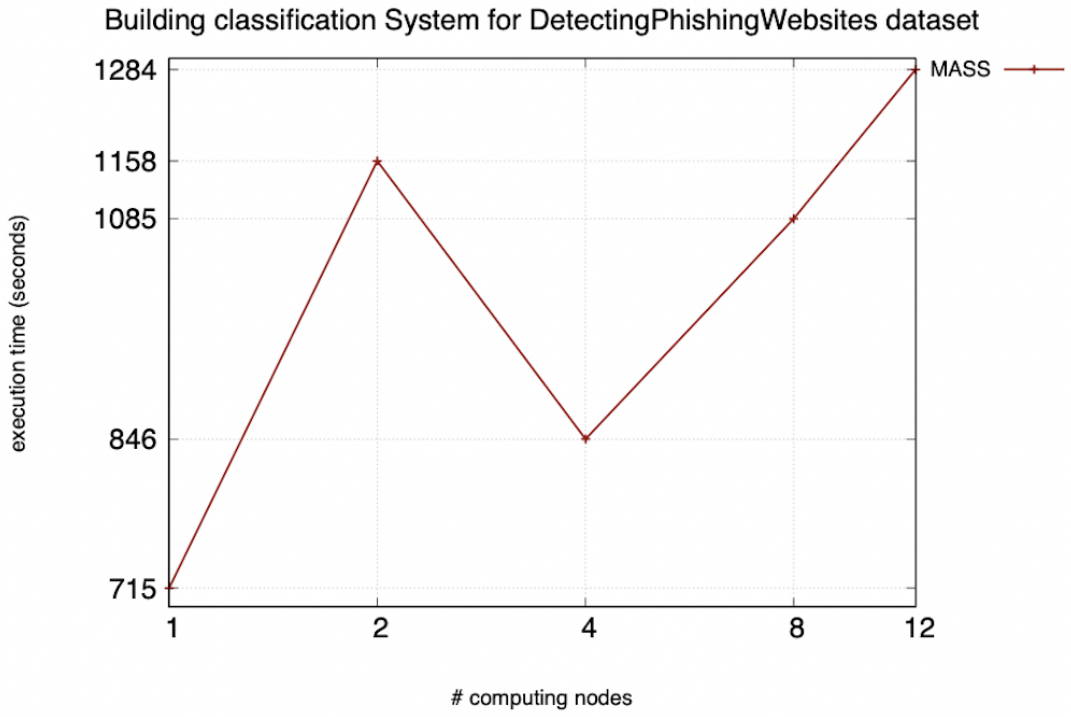


Figure 6: Scalability of GA using 6 Places and 6 Agents on 12 remote nodes

5.3 Bio-geography Based Optimization Performance

we have tested the algorithm with (3 jobs, 3 machines) , (6 jobs , 6 machines), and (20 jobs, 20 machines) and have included the readings for the largest job. we have gathered the test instances from Operations research website [13]. From Table 5 we can see that almost all the 3 implementation runtimes were very similar. But Repast Simphony had performed comparitively better for the given inputs.

Table 5: Time taken to execute 20 Jobs on 20 machines with 25 Agents

Time taken	**Serial Implementation**	**MASS Implementation**	**Repast Simphony Implementation**
for 1k iterations	292s	310s	273s
Best makespan	793	760	791

CPU scalability in MASS for BBO using callAll() on Agents

From Fig 7 we can see that though MASS has started from 310s, there was a significant decrease in the time taken to execute on multiple remote nodes. This can be because each Agent is working on 500 habitats each of size 20*20. If multiple Agents were used, then the time taken to execute might increase because of synchronization barrier between nodes.The best time taken to execute JSSP using BBO in MASS has produced best results on 24 nodes with 119s.

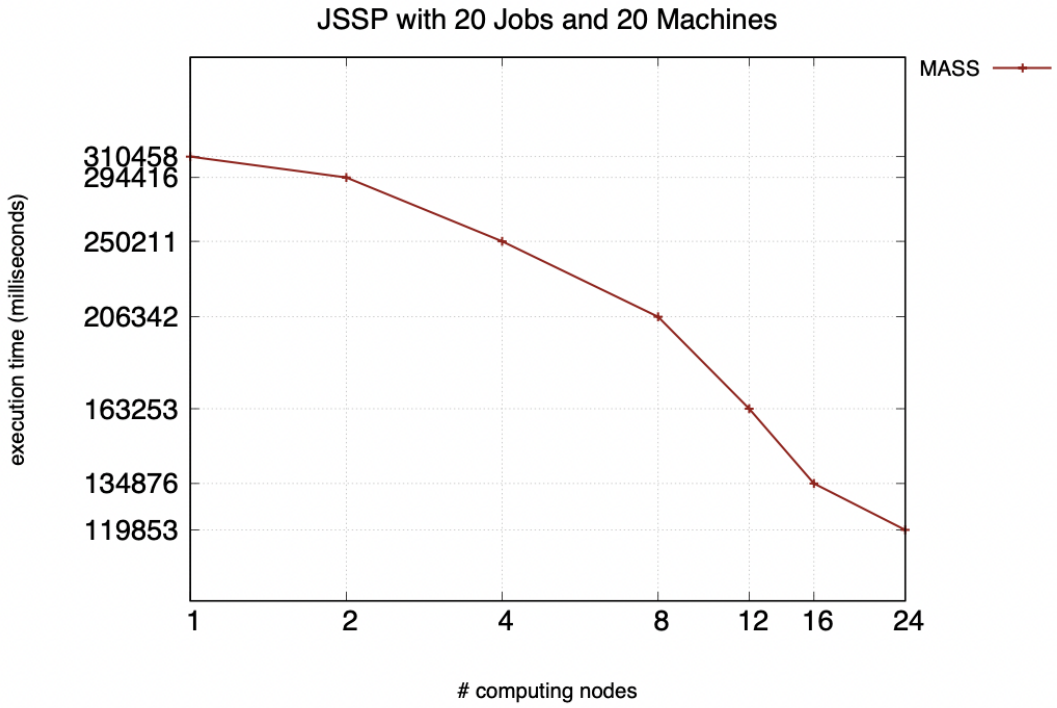


Figure 7: Scalability of BBO using 5 Places and 20 Agents over 24 remote nodes

6 Programmability Analysis

This section presents the Quantitative Analysis for some of the important components of the code that we have implemented. We will be comparing each of the 3 paradigms based on the number of classes occupied for each application, number of Lines of code required to implement the main logic of the applications and the Cyclomatic complexity of each application.

6.1 Quantitative Analysis of Classes

Table 6 depicts the number of classes required to implement applications using bio-inspired algorithms. Number of classes are higher compared to solely implementing the algorithms. This indicates that the process of developing end-to-end applications involves additional components and functionalities beyond the core algorithmic implementation. The increased number of classes reflects the complexity of the application development and highlights the comprehensive nature of incorporating bio-inspired algorithms into real world applications. Its interesting because though we have optimized the performance using Agent based modeling libraries the number of classes used still remains consistently similar. The minute difference between Serial and MASS implementation is because of extra classes were used to define the behavior of Places and Agents, the same applies for Repast Simphony where extra classes were used to Initialize the Context and define the behaviour of Agents.

Table 6: Number of classes for each implementation

Algorithm	Serial Implementation	MASS Implementation	Repast Simphony Implementation
ABC	12	12	12
GA	9	10	11
BBO	6	8	7

6.2 Quantitative Analysis of Lines of Code

Because MASS has the advantage of offering easy-to-use functionalities, Table 7 shows consistently fewer lines of code (LoC) was required for parallelization in MASS. With the use of

callAll() function, functionalities can be parallelized efficiently, saving time and effort compared to other libraries. Repast Simphony, although effective in enabling parallelization, presented challenges in terms of debugging when issues were encountered.

Table 7: LoC for Application Logic

LoC for	Serial Implementation	MASS Implementation	Repast Simphony Implementation
ABC	785	664	683
GA	630	579	549
BBO	573	510	546

6.3 Quantitative Analysis of Cyclomatic Complexity

Cyclomatic Complexity defines the complexity of a code block. If we observe Table 8, it depicts that MASS has shown slightly higher complexity than the other two implementations because of internal calls which were madeto MASS base functionalities.

Table 8: Cyclomatic Complexity

Cyclomatic complexity	Serial Implementation	MASS Implementation	Repast Simphony Implementation
ABC	3.965	4.04	3.999
GA	2.546	2.740	2.309
BBO	3.21	3.892	2.42

7 Conclusion & Future Work

Overall the project was successful in proving that Agent Based Modeling is a good paradigm to implement Bio-inspired Computing Algorithms like Artificial Bee Colony, Genetic Algorithm and Bio-geography based Optimization Algorithm. After implementing the algorithms in sequential paradigm and then implementing the same in MASS and Repast Simphony, it has been proved that MASS has performed better because of Agents powerful abilities to communicate with other agents present of multiple remote nodes. From 7 we can definitely say that MASS has better programmability and is more efficient than Repast Simphony.

Future Work :

1. **Explore the field of Bio-inspired Computing** : Bio-inspired Computing has a diverse set of algorithms out of which we cherry picked 3 algorithms to work on which helped us address some real world problems. Conducting a deep dive into this field might help us find some algorithms which will make the best use of Agent Behaviour in MASS. Ex: Tumor detection in brain images using Particle Swarm Optimization, Implement intelligent Tic-Tac-Toe game using Genetic Programming.

2. **Application Extensions**: Utilizing the capabilities of the algorithms we have currently developed, we can explore their applicability to solve more real-world problems.

3. **Decentralize data**: Due to the exploration and exploitation nature of Bio-inspired algorithms, we had to centralize the data in order for the agents to evolve. An in depth research can be conducted to find out if the data can be decentralized. This will help us make use of *doAll()* functionality and can reduce the synchronization overhead.

References

(1) Mert, U. Multi-Agent Spatial Simulation (MASS) Java Library Performance Improvement for Big Data Analysis, Ph.D. Thesis, University of Washington, 2017.

(2) Gordon, C.; Fukuda, M. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection: 16th International Conference, PAAMS 2018, Toledo, Spain, June 20–22, 2018, Proceedings 16*, 2018, pp 314–317.

(3) Torre-Bastida, A. I.; Dıaz-de-Arcaya, J.; Osaba, E.; Muhammad, K.; Camacho, D.; Del Ser, J. Bio-inspired computation for big data fusion, storage, processing, learning and visualization: state of the art and future directions. *Neural Computing and Applications* **2021**, 1–31.

(4) Fukuda, M. Mass: Parallel-computing library for multi-agent spatial simulation. *Distributed Systems Laboratory, Computing & Software Systems, University of Washington Bothell, Bothell, WA* **2010**.

(5) North, M. J.; Collier, N. T.; Ozik, J.; Tatara, E. R.; Macal, C. M.; Bragen, M.; Sydelko, P. Complex adaptive systems modeling with Repast Simphony. *Complex adaptive systems modeling* **2013**, *1*, 1–26.

(6) Ohlander, R.; Price, K.; Reddy, D. R. Picture segmentation using a recursive region splitting method. *Computer graphics and image processing* **1978**, *8*, 313–333.

(7) Karaboga, D. et al. *An idea based on honey bee swarm for numerical optimization*; tech. rep.; Technical report-tr06, Erciyes university, engineering faculty, computer . . ., 2005.

(8) Cuevas, E.; Sención, F.; Zaldivar, D.; Perez, M.; Sossa, H. A multi-threshold segmentation approach based on artificial bee colony optimization. *arXiv preprint arXiv:1405.7229* **2014**.

(9) Chaimontree, S.; Atkinson, K.; Coenen, F. In *Agents and Data Mining Interaction: 6th International Workshop on Agents and Data Mining Interaction, ADMI 2010, Toronto, ON, Canada, May 11, 2010, Revised Selected Papers 6*, 2010, pp 103–114.

(10) Pan, X.; Chen, H. In *2012 Eighth International Conference on Computational Intelligence and Security*, 2012, pp 123–127.

(11) Developers, G. Job Shop Scheduling, `https://developers.google.com/optimization/scheduling/job_shop`, Accessed: May 19, 2023.

(12) Wang, X.; Duan, H. A hybrid biogeography-based optimization algorithm for job shop scheduling problem. *Computers & Industrial Engineering* **2014**, *73*, 96–114.

(13) London, B. U. Job Shop Problem Instances, `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html`, Accessed: May 19, 2023.

8 Appendix

This section shows how we can execute Bio-inspired Computing algorithms on remote machines.

Fisrt pull applications from Srilekha-BioInspiredComputingAlgorithms branch.

we have already added required dependencies in the pom.xml file, please check for versions of dependencies being used and update accordingly in case of build failure.

path where we can find Bio-inspired computing algorithms :

```
[svrb@cssmpi1h ~]$ cd mass_java_appl/Benchmarks/
[svrb@cssmpi1h Benchmarks]$ ls
AgentBasedClassification          BreadthFirst     JobShopScheduling  ShortestPath       Triangles
BestTool                          ChinesePostman   KNN                SwarmOptimization  TrianglesDoAll
BioInspiredComputationAlgorithms  GradientDescent  PAAMS2021          TSP                pom.xml
[svrb@cssmpi1h Benchmarks]$
```

Artificial Bee Colony Alogorithm for Image Segmentation can be found in BioInspired-ComputationAlgorithms directory :

```
[svrb@cssmpi1h ~]$ cd mass_java_appl/Benchmarks/
[svrb@cssmpi1h Benchmarks]$ ls
AgentBasedClassification          BreadthFirst     JobShopScheduling  ShortestPath       Triangles
BestTool                          ChinesePostman   KNN                SwarmOptimization  TrianglesDoAll
BioInspiredComputationAlgorithms  GradientDescent  PAAMS2021          TSP                pom.xml
[svrb@cssmpi1h Benchmarks]$ cd BioInspiredComputationAlgorithms/
[svrb@cssmpi1h BioInspiredComputationAlgorithms]$ ls
logs  mass_fatal.log  mass_log.log  nodes.xml  pom.xml  resources  src  target
[svrb@cssmpi1h BioInspiredComputationAlgorithms]$
```

Build the package using mvn install & Build success message :

```
[WARNING] Artifact: edu.uwb.css534:mass-ABC:jar:1.0.0-SNAPSHOT references the same file as the assembly destination file. Moving i
t to a temporary location for inclusion.
[INFO] Building jar: /home/NETID/svrb/mass_java_appl/Benchmarks/BioInspiredComputationAlgorithms/target/mass-ABC-1.0.0-SNAPSHOT.ja
r
[WARNING] Configuration option 'appendAssemblyId' is set to false.
Instead of attaching the assembly file: /home/NETID/svrb/mass_java_appl/Benchmarks/BioInspiredComputationAlgorithms/target/mass-AB
C-1.0.0-SNAPSHOT.jar, it will become the file for main project artifact.
NOTE: If multiple descriptors or descriptor-formats are provided for this project, the value of this file will be non-deterministi
c!
[WARNING] Replacing pre-existing project main-artifact file: /home/NETID/svrb/mass_java_appl/Benchmarks/BioInspiredComputationAlgo
rithms/target/archive-tmp/mass-ABC-1.0.0-SNAPSHOT.jar
with assembly file: /home/NETID/svrb/mass_java_appl/Benchmarks/BioInspiredComputationAlgorithms/target/mass-ABC-1.0.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ mass-ABC ---
[INFO] Installing /home/NETID/svrb/mass_java_appl/Benchmarks/BioInspiredComputationAlgorithms/target/mass-ABC-1.0.0-SNAPSHOT.jar t
o /home/NETID/svrb/.m2/repository/edu/uwb/css534/mass-ABC/1.0.0-SNAPSHOT/mass-ABC-1.0.0-SNAPSHOT.jar
[INFO] Installing /home/NETID/svrb/mass_java_appl/Benchmarks/BioInspiredComputationAlgorithms/pom.xml to /home/NETID/svrb/.m2/repo
sitory/edu/uwb/css534/mass-ABC/1.0.0-SNAPSHOT/mass-ABC-1.0.0-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  40.347 s
[INFO] Finished at: 2023-06-05T15:02:27-07:00
[INFO] ------------------------------------------------------------------------
```

Run the ABC algorithm for Image Segmentation :

arg1 and arg2 determine the number of places and agents used to run the program and arg3 defines the number of iterations we want to run the program for and arg4 ask if we want to check the print messages.

```
[svrb@cssmpi1h BioInspiredComputationAlgorithms]$ java -jar target/mass-ABC-1.0.0-SNAPSHOT.jar 20 20 1000 y
size = 20 nAgents = 20
MProcess on cssmpi2h.uwb.edu run with command: java --add-modules java.se --add-exports java.base/jdk.internal.ref=ALL-UNNAMED --a
dd-opens java.base/java.lang=ALL-UNNAMED --add-opens java.base/java.nio=ALL-UNNAMED --add-opens java.base/sun.nio.ch=ALL-UNNAMED -
-add-opens java.management/sun.management=ALL-UNNAMED --add-opens jdk.management/com.sun.management.internal=ALL-UNNAMED -cp /home
/NETID/svrb/mass_java_appl/Benchmarks/BioInspiredComputationAlgorithms/target/*.jar edu.uw.bothell.css.dsl.MASS.MProcess HOSTNAME=
"cssmpi2h.uwb.edu" MYPID=1 NPROC=2 NTHREADS=1 SERVERPORT=28511 WORKDIR="/home/NETID/svrb/mass_java_appl/Benchmarks/BioInspiredComp
utationAlgorithms/target" MAXAGENTS=1000000 CLUSTERCOMMS="239.190.19.1:56313"
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.agrona.nio.TransportPoller (file:/home/NETID/svrb/mass_java_appl/Benchmarks/BioInspiredC
omputationAlgorithms/target/mass-ABC-1.0.0-SNAPSHOT.jar) to field sun.nio.ch.SelectorImpl.selectedKeys
WARNING: Please consider reporting this to the maintainers of org.agrona.nio.TransportPoller
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
MASS.init: done
going to initialize places now...
Did come here .. [0, 0]
Did come here .. [0, 1]
Did come here .. [0, 2]
Did come here .. [0, 3]
Did come here .. [0, 4]
Did come here .. [0, 5]
Did come here .. [0, 6]
Did come here .. [0, 7]
Did come here .. [0, 8]
Did come here .. [0, 9]
Did come here .. [0, 10]
Did come here .. [0, 11]
Did come here .. [0, 12]
Did come here .. [0, 13]
Did come here .. [0, 14]
Did come here .. [0, 15]
Did come here .. [0, 16]
Did come here .. [0, 17]
Did come here .. [0, 18]
Did come here .. [0, 19]
Did come here .. [1, 0]
Did come here .. [1, 1]



Did come here .. [8, 16]
Did come here .. [8, 17]
Did come here .. [8, 18]
Did come here .. [8, 19]
Did come here .. [9, 0]
Did come here .. [9, 1]
Did come here .. [9, 2]
Did come here .. [9, 3]
Did come here .. [9, 4]
Did come here .. [9, 5]
Did come here .. [9, 6]
Did come here .. [9, 7]
Did come here .. [9, 8]
Did come here .. [9, 9]
Did come here .. [9, 10]
Did come here .. [9, 11]
Did come here .. [9, 12]
Did come here .. [9, 13]
Did come here .. [9, 14]
Did come here .. [9, 15]
Did come here .. [9, 16]
Did come here .. [9, 17]
Did come here .. [9, 18]
Did come here .. [9, 19]
place initialization DONE...
Go!
... step 1: Started ABC implementation for iteration.................
Total time taken for MASS iterations: 35817
****************** MASS iterations are done ******************
Best solution: [0.33944499492645264, 160.0, 2072.0, 0.21833600103855133, 189.0, 18124.0, 0.4422190189361572, 188.0, 36586.0]
{160.0=0, 188.0=2, 189.0=1}
[0.33944499492645264, 160.0, 2072.0, 0.21833600103855133, 189.0, 18124.0, 0.4422190189361572, 188.0, 36586.0]
[0.33944499492645264, 160.0, 2072.0, 0.4422190189361572, 188.0, 36586.0, 0.21833600103855133, 189.0, 18124.0]
[17960.484062184216, 3507827.930890417]
MASS Shutting Down...
Sending shutdown request to cssmpi2h.uwb.edu
MASS Shutdown Finished
[svrb@cssmpi1h BioInspiredComputationAlgorithms]$
```

we can find the inputs and outputs produced by the application in resources directory :

output images will have segmented output tag atatched to the input.

```
... step 1: Started ABC implementation for iteration.................
Total time taken for MASS iterations: 35817
****************** MASS iterations are done ********************
Best solution: [0.33944499492645264, 160.0, 2072.0, 0.21833600103855133, 189.0, 18124.0, 0.4422190189361572, 188.0, 36586.0]
{160.0=0, 188.0=2, 189.0=1}
[0.33944499492645264, 160.0, 2072.0, 0.21833600103855133, 189.0, 18124.0, 0.4422190189361572, 188.0, 36586.0]
[0.33944499492645264, 160.0, 2072.0, 0.4422190189361572, 188.0, 36586.0, 0.21833600103855133, 189.0, 18124.0]
[17960.484062184216, 3507827.930890417]
MASS Shutting Down...
Sending shutdown request to cssmpi2h.uwb.edu
MASS Shutdown Finished
[svrb@cssmpi1h BioInspiredComputationAlgorithms]$ cd resources/
[svrb@cssmpi1h resources]$ ls
brain_398.tif                                          fmt_galaxyforce2_0001_front_SegmentedOutput_1684427225488.tif
brain_398_grayscale_.tif                               fmt_galaxyforce2_0001_front_SegmentedOutput_1684427893132.tif
brain_508.tif                                          fmt_galaxyforce2_0001_front_SegmentedOutput_1684428624085.tif
brain_508_grayscale_.tif                               fmt_galaxyforce2_0001_front_SegmentedOutput_1684429172681.tif
chest-image.tif                                        fmt_galaxyforce2_0001_front_SegmentedOutput_1686002784242.tif
fmt_galaxyforce2_0001_front.tif                        fmt_galaxyforce2_0001_front_grayscale_.tif
fmt_galaxyforce2_0001_front_SegmentedOutput_1684426640441.tif  m83.tif
[svrb@cssmpi1h resources]$
```

Genetic Alogorithm for Building Classification System can be found in AgentBasedClassification directory :

```
[svrb@cssmpi1h Benchmarks]$ ls
AgentBasedClassification          BreadthFirst     JobShopScheduling  ShortestPath       Triangles
BestTool                          ChinesePostman   KNN                SwarmOptimization  TrianglesDoAll
BioInspiredComputationAlgorithms  GradientDescent  PAAMS2021          TSP                pom.xml
[svrb@cssmpi1h Benchmarks]$ cd AgentBasedClassification
[svrb@cssmpi1h AgentBasedClassification]$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< edu.uwb.css534:mass-AgentBasedClassification >------------
[INFO] Building MASS-AgentBasedClassification 1.0.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
Downloading from central: https://repo.maven.apache.org/maven2/be/abeel/ajt/2.9/ajt-2.9.pom
Downloading from mvn: https://bio.informatik.uni-jena.de/repository/libs-release-oss/be/abeel/ajt/2.9/ajt-2.9.pom
Downloading from neu: https://neuroph.sourceforge.net/maven2/be/abeel/ajt/2.9/ajt-2.9.pom
Downloading from jboss: https://repository.jboss.org/maven2/be/abeel/ajt/2.9/ajt-2.9.pom
[WARNING] The POM for be.abeel:ajt:jar:2.9 is missing, no dependency information available
Downloading from mvn: https://bio.informatik.uni-jena.de/repository/libs-release-oss/org/agrona/agrona/maven-metadata.xml
Downloading from neu: https://neuroph.sourceforge.net/maven2/org/agrona/agrona/maven-metadata.xml
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ mass-AgentBasedClassification ---
[WARNING] Using platform encoding (ANSI_X3.4-1968 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ mass-AgentBasedClassification ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 11 source files to /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ mass-AgentBasedClassification ---
[WARNING] Using platform encoding (ANSI_X3.4-1968 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ mass-AgentBasedClassification ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ mass-AgentBasedClassification ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ mass-AgentBasedClassification ---
```

to build the package use mvn install command :

```
[svrb@cssmpi1h Benchmarks]$ ls
AgentBasedClassification          BreadthFirst     JobShopScheduling  ShortestPath       Triangles
BestTool                          ChinesePostman   KNN                SwarmOptimization  TrianglesDoAll
BioInspiredComputationAlgorithms  GradientDescent  PAAMS2021          TSP                pom.xml
[svrb@cssmpi1h Benchmarks]$ cd AgentBasedClassification
[svrb@cssmpi1h AgentBasedClassification]$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< edu.uwb.css534:mass-AgentBasedClassification >------------
[INFO] Building MASS-AgentBasedClassification 1.0.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
Downloading from central: https://repo.maven.apache.org/maven2/be/abeel/ajt/2.9/ajt-2.9.pom
Downloading from mvn: https://bio.informatik.uni-jena.de/repository/libs-release-oss/be/abeel/ajt/2.9/ajt-2.9.pom
Downloading from neu: https://neuroph.sourceforge.net/maven2/be/abeel/ajt/2.9/ajt-2.9.pom
Downloading from jboss: https://repository.jboss.org/maven2/be/abeel/ajt/2.9/ajt-2.9.pom
[WARNING] The POM for be.abeel:ajt:jar:2.9 is missing, no dependency information available
Downloading from mvn: https://bio.informatik.uni-jena.de/repository/libs-release-oss/org/agrona/agrona/maven-metadata.xml
Downloading from neu: https://neuroph.sourceforge.net/maven2/org/agrona/agrona/maven-metadata.xml
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ mass-AgentBasedClassification ---
[WARNING] Using platform encoding (ANSI_X3.4-1968 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ mass-AgentBasedClassification ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 11 source files to /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ mass-AgentBasedClassification ---
[WARNING] Using platform encoding (ANSI_X3.4-1968 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ mass-AgentBasedClassification ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ mass-AgentBasedClassification ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ mass-AgentBasedClassification ---
```

```
[WARNING] Configuration option 'appendAssemblyId' is set to false.
Instead of attaching the assembly file: /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/target/mass-AgentBased
Classification-1.0.0-SNAPSHOT.jar, it will become the file for main project artifact.
NOTE: If multiple descriptors or descriptor-formats are provided for this project, the value of this file will be non-deterministi
c!
[WARNING] Replacing pre-existing project main-artifact file: /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/t
arget/archive-tmp/mass-AgentBasedClassification-1.0.0-SNAPSHOT.jar
with assembly file: /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/target/mass-AgentBasedClassification-1.0.0
-SNAPSHOT.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ mass-AgentBasedClassification ---
[INFO] Installing /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/target/mass-AgentBasedClassification-1.0.0-S
NAPSHOT.jar to /home/NETID/svrb/.m2/repository/edu/uwb/css534/mass-AgentBasedClassification/1.0.0-SNAPSHOT/mass-AgentBasedClassifi
cation-1.0.0-SNAPSHOT.jar
[INFO] Installing /home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/pom.xml to /home/NETID/svrb/.m2/repository/e
du/uwb/css534/mass-AgentBasedClassification/1.0.0-SNAPSHOT/mass-AgentBasedClassification-1.0.0-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  23.933 s
[INFO] Finished at: 2023-06-05T15:10:36-07:00
[INFO] ------------------------------------------------------------------------
```

run the GA algorithm using 6places and 6 agents for 1k iterations :

```
[svrb@cssmpi1h AgentBasedClassification]$ java -jar target/mass-AgentBasedClassification-1.0.0-SNAPSHOT.jar 6 6 1000 y
phishingData.data
/home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/phishingData.data
size = 6 nAgents = 6
MProcess on cssmpi2h.uwb.edu run with command: java --add-modules java.se --add-exports java.base/jdk.internal.ref=ALL-UNNAMED --a
dd-opens java.base/java.lang=ALL-UNNAMED --add-opens java.base/java.nio=ALL-UNNAMED --add-opens java.base/sun.nio.ch=ALL-UNNAMED -
-add-opens java.management/sun.management=ALL-UNNAMED --add-opens jdk.management/com.sun.management.internal=ALL-UNNAMED -cp /home
/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassification/target/*.jar edu.uw.bothell.css.dsl.MASS.MProcess HOSTNAME="cssmpi2
h.uwb.edu" MYPID=1 NPROC=2 NTHREADS=1 SERVERPORT=28511 WORKDIR="/home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedClassificatio
n/target" MAXAGENTS=1000000 CLUSTERCOMMS="239.230.196.1:48558"
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.agrona.nio.TransportPoller (file:/home/NETID/svrb/mass_java_appl/Benchmarks/AgentBasedCl
assification/target/mass-AgentBasedClassification-1.0.0-SNAPSHOT.jar) to field sun.nio.ch.SelectorImpl.selectedKeys
WARNING: Please consider reporting this to the maintainers of org.agrona.nio.TransportPoller
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
MASS.init: done
going to initialize places now...
Did come here .. [0, 0]
Did come here .. [0, 1]
Did come here .. [0, 2]
Did come here .. [0, 3]
Did come here .. [0, 4]
Did come here .. [0, 5]
Did come here .. [1, 0]
Did come here .. [1, 1]
Did come here .. [1, 2]
Did come here .. [1, 3]
Did come here .. [1, 4]
Did come here .. [1, 5]
Did come here .. [2, 0]
Did come here .. [2, 1]
Did come here .. [2, 2]
Did come here .. [2, 3]
Did come here .. [2, 4]
Did come here .. [2, 5]
place initialization DONE...
Go!
```

```
MASS.init: done
going to initialize places now...
Did come here .. [0, 0]
Did come here .. [0, 1]
Did come here .. [0, 2]
Did come here .. [0, 3]
Did come here .. [0, 4]
Did come here .. [0, 5]
Did come here .. [1, 0]
Did come here .. [1, 1]
Did come here .. [1, 2]
Did come here .. [1, 3]
Did come here .. [1, 4]
Did come here .. [1, 5]
Did come here .. [2, 0]
Did come here .. [2, 1]
Did come here .. [2, 2]
Did come here .. [2, 3]
Did come here .. [2, 4]
Did come here .. [2, 5]
place initialization DONE...
Go!
MASS Shutting Down...
Sending shutdown request to cssmpi2h.uwb.edu
MASS Shutdown Finished
Final accuracy: 0.8850678733031674
TOTAL TIME TAKEN: 740277
[svrb@cssmpi1h AgentBasedClassification]$
```

Bio-geography Based Optimization Algorithm for Implementing Job Shop Scheduling Problem can be found in JobShopScheduling directory :

```
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ mass-JobShopScheduling ---
[INFO] Building jar: /home/NETID/svrb/mass_java_appl/Benchmarks/JobShopScheduling/target/mass-JobShopScheduling-1.0.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-assembly-plugin:3.1.0:single (default) @ mass-JobShopScheduling ---
[WARNING] Missing POM for be.abeel:ajt:jar:2.9
Downloading from netbeans: http://bits.netbeans.org/nexus/content/groups/netbeans/edu/ucar/netcdf4/5.1.0/netcdf4-5.1.0.pom
Downloading from netbeans: http://bits.netbeans.org/nexus/content/groups/netbeans/edu/ucar/cdm/5.1.0/cdm-5.1.0.pom
Downloading from netbeans: http://bits.netbeans.org/nexus/content/groups/netbeans/edu/ucar/udunits/5.1.0/udunits-5.1.0.pom
Downloading from netbeans: http://bits.netbeans.org/nexus/content/groups/netbeans/edu/ucar/httpservices/5.1.0/httpservices-5.1.0.p
om
[WARNING] Artifact: edu.uwb.css534:mass-JobShopScheduling:jar:1.0.0-SNAPSHOT references the same file as the assembly destination
file. Moving it to a temporary location for inclusion.
[INFO] Building jar: /home/NETID/svrb/mass_java_appl/Benchmarks/JobShopScheduling/target/mass-JobShopScheduling-1.0.0-SNAPSHOT.jar
[WARNING] Configuration option 'appendAssemblyId' is set to false.
Instead of attaching the assembly file: /home/NETID/svrb/mass_java_appl/Benchmarks/JobShopScheduling/target/mass-JobShopScheduling
-1.0.0-SNAPSHOT.jar, it will become the file for main project artifact.
NOTE: If multiple descriptors or descriptor-formats are provided for this project, the value of this file will be non-deterministi
c!
[WARNING] Replacing pre-existing project main-artifact file: /home/NETID/svrb/mass_java_appl/Benchmarks/JobShopScheduling/target/a
rchive-tmp/mass-JobShopScheduling-1.0.0-SNAPSHOT.jar
with assembly file: /home/NETID/svrb/mass_java_appl/Benchmarks/JobShopScheduling/target/mass-JobShopScheduling-1.0.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  17.644 s
[INFO] Finished at: 2023-06-05T15:47:07-07:00
[INFO] ------------------------------------------------------------------------
[svrb@cssmpi1h JobShopScheduling]$
```

run the JSSP algorithm using 5 places and 20 agents for 1k iterations :

```
[svrb@cssmpi1h JobShopScheduling]$ java -jar target/mass-JobShopScheduling-1.0.0-SNAPSHOT.jar 5 20 1000 y
MASS implementation of JSSP starting...
size = 5 nAgents = 20
MProcess on cssmpi2h.uwb.edu run with command: java --add-modules java.se --add-exports java.base/jdk.internal.ref=ALL-UNNAMED --a
dd-opens java.base/java.lang=ALL-UNNAMED --add-opens java.base/java.nio=ALL-UNNAMED --add-opens java.base/sun.nio.ch=ALL-UNNAMED -
-add-opens java.management/sun.management=ALL-UNNAMED --add-opens jdk.management/com.sun.management.internal=ALL-UNNAMED -cp /home
/NETID/svrb/mass_java_appl/Benchmarks/JobShopScheduling/target/*.jar edu.uw.bothell.css.dsl.MASS.MProcess HOSTNAME="cssmpi2h.uwb.e
du" MYPID=1 NPROC=2 NTHREADS=1 SERVERPORT=28511 WORKDIR="/home/NETID/svrb/mass_java_appl/Benchmarks/JobShopScheduling/target" MAXA
GENTS=1000000 CLUSTERCOMMS="239.237.169.1:54721"
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.agrona.nio.TransportPoller (file:/home/NETID/svrb/mass_java_appl/Benchmarks/JobShopSched
uling/target/mass-JobShopScheduling-1.0.0-SNAPSHOT.jar) to field sun.nio.ch.SelectorImpl.selectedKeys
WARNING: Please consider reporting this to the maintainers of org.agrona.nio.TransportPoller
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
MASS.init: done
going to initialize places now...
Did come here .. [0, 0]
Did come here .. [0, 1]
Did come here .. [0, 2]
Did come here .. [0, 3]
Did come here .. [0, 4]
Did come here .. [1, 0]
Did come here .. [1, 1]
Did come here .. [1, 2]
Did come here .. [1, 3]
Did come here .. [1, 4]
Did come here .. [2, 0]
Did come here .. [2, 1]
place initialization DONE...
MASS Shutting Down...
Sending shutdown request to cssmpi2h.uwb.edu
MASS Shutdown Finished
total time: 3863
Best HSI: 35
Best Habitat: [1, 2, 3, 3, 1, 2, 2, 3, 1]
[svrb@cssmpi1h JobShopScheduling]$
```

For visual representation of how algorithms work, refer to SrilekhaBandaru-FinalDefensePresentation