FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION FOR HIGHER PROFESSIONAL EDUCATION NATIONAL RESEARCH UNIVERSITY

«HIGHER SCHOOL OF ECONOMICS»

Faculty of Computer Science

Abhishek Kardam

Генетический алгоритм, основанный на кластеризации, в решении задач комбинаторной оптимизации

Clustering based Genetic Algorithm in Addressing Combinatorial Optimization Problems

Qualification paper – Master of Science Dissertation

Field of study 01.04.02 «Applied Mathematics and Informatics»

Program: Data Science

Student

Abhishek Kardam

Supervisor

Majid Sohrabi

Assistant, Research Assistant,

Department of Data Analysis and Artificial Intelligence

Moscow, 2024

Table of Content

Table of Abbreviation

Genetic Algorithm	GA
Density-Based Spatial Clustering of Applications with Noise	DBSCAN
Affinity Propagation	AP
Mean Shift	MS
Formal Concept Analysis	FCA

ABSTRACT

The combination of clustering methods with Genetic Algorithms (GAs) for addressing the Knapsack problem results in considerable increases in optimization efficiency and solution quality. This study investigates many hybrid models, including the usage of DBSCAN, K-means, Affinity Propagation, and Mean Shift clustering algorithms mixed with GAs. Using clustering approaches, the solution space is efficiently partitioned, which improves the evolutionary search process. The Genetic Algorithm with Formal Concept Analysis (FCA)-based Clustering is a revolutionary technique in this work, since it refines the clustering phase using FCA principles. This approach not only finds noteworthy patterns in the data, but it also enhances the whole optimization process by providing a systematic framework for conceptual clustering of solutions.

The suggested hybrid algorithms were tested in a number of experiments, proving their capacity to balance exploration and exploitation in the search space. The results show that clustering-based GAs outperform standard GAs in terms of convergence rates and solution quality. Specifically, the FCA-based clustering strategy performed well in big datasets and challenging optimization problems by preserving genetic diversity and preventing premature convergence. The combination of GAs and sophisticated clustering techniques yields a stable and scalable solution to the Knapsack problem and other combinatorial optimization problems.

Keywords

Genetic algorithms, Formal Concept Analysis, Knapsack Problem, DBSCAN Clustering, Mean Shift Clustering, Combinatorial Optimization Problem.

Аннотация

Для решения задачи о рюкзаке методы кластеризации с генетическими алгоритмами (ГА) значительно улучшают эффективность оптимизации и качество решений. В этом исследовании рассматриваются различные гибридные модели, включая алгоритмы кластеризации DBSCAN, K-means, Affinity Propagation и Mean Shift в сочетании с GA. Методы кластеризации улучшают эволюционный процесс поиска, разделяя пространство решений. В этом исследовании выделяется новый подход, генетический алгоритм с кластеризацией, основанный на анализе формальных понятий (АФП). Он использует принципы АФП для уточнения этапа кластеризации. Этот метод не только помогает найти важные паттерны в данных, но и улучшает в целом процесс оптимизации, предоставляя структурированную основу для концептуальной кластеризации решений.

Серия экспериментов была проведена для оценки предложенных гибридных алгоритмов; они показали, что они способны сбалансировать исследование и использование в пространстве поиска. Результаты показывают, что кластеризация с ГА могут сравиться с традиционными ГА с более высоким уровнем сходимости и качеством решений. В частности, метод кластеризации на основе АФП показал отличные результаты в оптимизации сложных задач и обработке больших наборов данных. Кроме того, он поддерживает генетическое разнообразие и предотвращает преждевременную сходимость. Таким образом, для решения задачи о рюкзаке и подобных комбинаторных оптимизационных задач сочетание ГА с продвинутыми методами кластеризации дает масштабируемое и надежное решение.

1. Introduction

In the realm of optimization challenges, combinatorial optimization issues stand out. These problems involve selecting the best solution from a finite set of options. They permeate numerous fields, from engineering and logistics to finance and bioinformatics. However, a significant hurdle in combinatorial optimization is the exponential growth in the number of potential solutions as the problem size increases. This complexity makes classical optimization methods impractical for large-scale problems.

Genetic Algorithms (GA) are powerful tools that draw inspiration from the principles of natural selection and genetics. Since their introduction by Holland in the 1970s, GAs have been extensively researched and applied across various optimization challenges. Their strength lies in their ability to rapidly explore the solution space, often homing in on optimal or near-optimal solutions, even in complex, high-dimensional search spaces.

Clustering algorithms, on the other hand, are crucial for dividing data into groups based on similarity or proximity. These algorithms find applications in consumer segmentation, anomaly detection, and pattern recognition, and more. In the context of combinatorial optimization, clustering can significantly enhance the performance of optimization algorithms like GAs by effectively partitioning the solution space into meaningful subsets.

Among the array of clustering algorithms, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) stands out. It excels in identifying clusters of varying shapes within noisy data by clustering tightly packed points and labeling points in less dense regions as outliers. This capability makes the algorithm ideal for datasets with irregular cluster shapes and densities.

In contrast, K-Means clustering, a well-known centroid-based technique, segments data into a predetermined number of clusters. It iteratively assigns data points to the nearest centroid and updates the centroids based on the mean of the assigned points. Despite its simplicity, K-Means is both efficient and effective, especially when the data is well-separated and the clusters are roughly spherical.

Mean-Shift clustering offers a different approach. It’s a non-parametric method that doesn’t require prior knowledge of the number of clusters. Mean-Shift operates by iteratively shifting each data point towards the mode of the data distribution, thereby identifying clusters as regions of high data density. Its resistance to noise and ability to handle irregularly shaped clusters add to its versatility.

Another innovative method is Affinity Propagation clustering. This technique allows data points to communicate with one another to identify exemplars, or representative points of clusters. Through message passing between data points, clusters and their exemplars emerge. Affinity Propagation is particularly adept at recognizing clusters of various sizes and densities.

Formal Concept Analysis (FCA) provides a mathematical framework for conceptual clustering. It aims to extract meaningful concepts from data by identifying groups of objects with similar attributes. FCA helps uncover underlying structures in complex datasets, facilitating the discovery of significant patterns.

Combining genetic algorithms with clustering techniques like DBSCAN, K-Means, Mean-Shift, Affinity Propagation, and FCA can harness the strengths of both optimization and clustering. This hybrid approach allows for efficient exploration of the solution space, guided by the structures and features identified during clustering. By integrating these techniques, we can enhance the scalability, convergence, and solution quality of combinatorial optimization problems, such as the Knapsack problem.

The goal of this research is to investigate the synergies between GA, clustering algorithms, and FCA in the context of combinatorial optimization problems, with a special emphasis on the Knapsack problem. The Knapsack problem is a classic optimization task in which the aim is to maximize the value of things chosen for inclusion in a bag while keeping the overall weight of the bag under a certain limit. Finding an ideal solution gets more difficult when the number of viable answers grows exponentially.

The study is divided into sections to methodically analyze and evaluate the effectiveness of various techniques. The introduction gives a thorough review of combinatorial optimization difficulties, the relevance of GAs, and the importance of clustering algorithms and FCA in improving optimization strategies. Following the introduction, the literature review(Section 2) digs into previous research on GAs, clustering algorithms, FCA, and their applications in combinatorial optimization, laying the theoretical groundwork for the proposed technique.

The next parts describe the problem statement (Section 3), data preparation (Section 3.1), and proposed approaches (Section 4), which include genetic algorithms both alone and in conjunction with various clustering techniques. The experimental results (Section 5) are provided and evaluated in depth to assess the performance of each strategy in terms of solution quality, convergence speed, and scalability. The conclusion (Section 6) summarizes the findings and suggests potential areas of further investigation. Overall, the work intends to develop optimization approaches by combining the combined characteristics of GAs, clustering algorithms, and FCA to handle combinatorial optimization difficulties.

2. Literature Review

Combinatorial optimization issues are common in a variety of domains, including logistics, finance, engineering, and bioinformatics. These challenges require choosing the optimal solution from a limited number of possibilities, frequently under limits on resources or objectives. Khuri, Bäck, and Heitkötter (1994) proposed an evolutionary approach to combinatorial optimization problems, highlighting the usefulness of metaheuristic approaches such as genetic algorithms (GAs). Their work paved the way for the efficient application of evolutionary ideas to complicated optimization problems.

Misevičius, Blažauskas, Blonskis, and Smolinskas (2004) provided a detailed survey of heuristic algorithms used in combinatorial optimization. They explored the uses and limits of various heuristic methods, emphasizing the importance of robust optimization strategies for efficiently addressing real-world situations. This study provides useful insights into the landscape of combinatorial optimization algorithms, which will guide future research in the subject.

Genetic algorithms (GAs) are strong optimization techniques based on natural selection and genetic principles. Harik and Lobo (1999) introduced a parameter-less genetic algorithm with the goal of eliminating the requirement for human parameter adjustment, which may be time-consuming and complicated. Their technique centered on self-adaptation, which enabled the algorithm to dynamically modify its parameters during the optimization process. This parameter-less GA provided an effective approach for optimizing a variety of combinatorial problems.

In recent years, Sohrabi, Fathollahi-Fard, and Gromov (2024) presented the Genetic Engineering Algorithm (GEA), stressing its effectiveness in handling combinatorial optimization issues. GEA combines unique genetic operators and selection methods, which improve the algorithm's exploration and exploitation capabilities. The study confirmed GEA's efficiency in a variety of optimization tasks, establishing it as a potential metaheuristic approach in evolutionary computation.

Haldurai, Madhubala, and Rajalakshmi (2016) performed a thorough investigation on genetic algorithms and their use in optimization. Their findings provide light on GA processes like selection, crossover, and mutation, revealing the importance of each component in the optimization process. By examining several GA applications in diverse disciplines, the study demonstrated the adaptability and efficacy of genetic algorithms in tackling complicated optimization issues.

Clustering algorithms are essential for categorizing data into meaningful groupings based on similarity criteria. Wang, Zhang, Li, Zhang, and Guo (2008) proposed Adaptive Affinity Propagation Clustering, which improves the flexibility of conventional affinity propagation techniques. Their technique constantly modifies the clustering parameters based on the properties of the input data, increasing the clustering algorithm's resilience and scalability. This adaptive clustering approach has applications in a variety of domains, including pattern detection and data mining.

Creţulescu, Morariu, Breazu, and Volovici (2019) investigated the use of the DBSCAN algorithm in document clustering, emphasizing its usefulness in dealing with noisy data.
Likas, Vlassis, and Verbeek (2003) developed the Global K-Means Clustering method, which tries to divide data into a predefined number of clusters effectively. Unlike standard K-Means algorithms, which may converge to poor solutions depending on initial centroid placements, Global K-Means improves clustering performance using a global optimization approach. This approach has applications in picture segmentation, pattern recognition, and data analysis, where precise clustering is required for subsequent decision-making procedures.

Formal Concept Analysis (FCA) is a mathematical approach for conceptual clustering that seeks to extract meaningful concepts from data. Hristoskova, Boeva, and Tsiporkova (2014) used FCA for consensus clustering of multi-experiment expression data, proving its applicability to biological data processing. FCA allows researchers to better comprehend complicated biological systems by recognizing common patterns across repeated tests.

Sumangali and Ch (2019) presented a concept lattice simplification approach based on attribute clustering to improve the interpretability of formal concept lattices. Their technique tries to simplify idea lattices by grouping comparable features together, resulting in simple and useful representations of data connections. This strategy increases the efficiency of knowledge discovery procedures in a variety of disciplines, including data mining and information retrieval, by simplifying idea lattices.

The combination of evolutionary algorithms, clustering algorithms, and FCA provides a synergistic method for solving difficult problems effectively. Chehouri, Younes, Khoder, Perron, and Ilinca (2017) suggested a clustering-based selection approach for genetic algorithms to increase the variety of the algorithm's produced solutions. By adding clustering insights into the selection process, their method enhances the exploration-exploitation balance of genetic algorithms, resulting in improved optimization efficiency.

Zeebaree, Haron, Abdulazeez, and Zeebaree (2017) performed a review on the combination of K-Means clustering and evolutionary algorithms, emphasizing its potential for solving complicated issues. This hybrid technique increases genetic algorithm convergence speed and solution quality by combining clustering-based initialization with genetic operators. Feng (2012) investigated data clustering with genetic algorithms, highlighting the synergy between the optimization and clustering paradigms. This integration allows for fast exploration of solution spaces, led by clustering insights, resulting in better optimization performance.

Previous research has mostly focused on Genetic techniques (GAs), either alone or in conjunction with clustering techniques, while ignoring the potential benefits of incorporating Formal Concept Analysis (FCA) into the optimization framework. The absence of investigation of alternate methodologies, such as FCA, has led to significant gaps in the research. As a result, this study attempts to identify and fix these gaps by using the complementing properties of GAs, clustering techniques, and FCA. Through this combination, the research seeks to give a complete and efficient framework for tackling complicated optimization issues that were previously difficult to address successfully.

One notable gap is the poor scalability and convergence of current optimization approaches, especially when dealing with large-scale combinatorial optimization problems with different solution spaces. While GAs and clustering algorithms have proven useful in specific situations, they frequently fail to quickly explore large solution spaces and uncover significant patterns in high-dimensional data sets. Furthermore, present techniques may face difficulties in properly balancing exploration and exploitation to reach optimum solutions within realistic computing timescales.

By including FCA into the optimization framework, this study hopes to fill these gaps and open up new paths for tackling complicated optimization issues. FCA provides a formal mathematical framework for conceptual clustering, which improves the interpretability and efficiency of optimization algorithms by disclosing underlying solution structures. By introducing FCA-based clustering insights into the optimization process, the proposed method aims to increase scalability, convergence, and solution quality.

Furthermore, by combining Formal Concept Analysis (FCA) with Genetic Algorithms (GAs) and clustering algorithms, this work intends to close the scalability, convergence, and interpretability gaps discovered in prior studies of optimization techniques. Using FCA's capacity to extract meaningful concepts from data, the suggested integration provides a fresh way for dealing with difficult combinatorial optimization issues. By introducing FCA-based clustering insights into the optimization process, the study aims to improve the efficiency of exploration and exploitation tactics, ultimately boosting solution quality and computing scalability. This holistic approach not only increases the state-of-the-art in optimization approaches, but it also provides a complete framework for addressing real-world difficulties across several domains.

3. Problem Statement

A famous optimization issue is the "knapsack problem," which is figuring out the optimum way to fill a knapsack with stuff while staying within its weight limit. In other words, given a collection of objects, each with its own weight and worth, the aim is to maximize the overall value of the items in the knapsack while staying within the weight restriction. This problem arises in a variety of real-world situations, including resource allocation, scheduling, and portfolio optimization.

The Knapsack problem has been the topic of substantial research in combinatorial optimization, with multiple publications suggesting various algorithms and strategies to overcome its obstacles. Khuri, Bäck, and Heitkötter (1994) proposed an evolutionary method to tackling combinatorial optimization issues, including the Knapsack problem, emphasizing the ability of metaheuristic algorithms such as Genetic Algorithms (GAs) to effectively explore the solution space.

Pisinger (1999) outlined basic difficulties in Knapsack algorithms, highlighting the importance of good optimization strategies to solve the complexity of large-scale Knapsack cases. Furthermore, Caprara, Pisinger, and Toth (1999) proposed an exact solution approach for the quadratic Knapsack problem, proving the viability of obtaining optimum solutions using rigorous mathematical techniques.

Cacchiani et al. (2022) reviewed recent improvements in Knapsack issues, concentrating on multiple, multidimensional, and quadratic Knapsack variants. Their research emphasizes on the increased interest in building efficient algorithms to solve distinct Knapsack problem formulations across several application domains.

Formally, the Knapsack problem is defined as follows: Let n be the number of items available for selection, and w_i and v_i represent the weight and value of item i, where i ranges from 1 to n. Additionally, let W indicate the knapsack's maximum weight capacity. The goal is to identify a subset (S) of items with a total weight of less than or equal to W and a maximum total value.

In mathematics, this may be expressed as:

$$maximize \sum_{i=1}^{n} v_i \cdot x_i \qquad (1)$$

$$subject\ to \sum_{i=1}^{n} w_i \cdot x_i \leq W \tag{2}$$

$$where\ x_i \in \{0, 1\}\ for\ i = 1, 2, \ldots, n \tag{3}$$

Here, x_i is a binary variable that indicates whether item i is picked (1 or 0). The goal function optimizes the overall worth of the selected goods while ensuring that the total weight does not exceed the capacity of the knapsack.

The Knapsack issue is a typical example of a combinatorial optimization problem since it requires choosing a combination of items from a discrete set of options in order to maximize a given objective function. In this scenario, the discrete options are whether or not to include each item in the knapsack, and the goal is to maximize the overall value of the selected items while adhering to the weight limit.

To solve the Knapsack problem, a variety of ways have been developed, including exact algorithms, heuristic and metaheuristic strategies. Exact algorithms seek the best answer by thoroughly examining all potential combinations of things, which can be computationally expensive for complex problem situations. Heuristic algorithms, on the other hand, give approximate answers by using intuitive tactics to swiftly locate excellent solutions, however they do not guarantee optimality.

GAs have been frequently employed to solve Knapsack issues because of their capacity to effectively explore huge solution spaces and deal with discrete decision factors. GAs use evolutionary concepts such as selection, crossover, and mutation to iteratively improve the quality of candidate solutions across successive generations, portraying them as strings of binary digits. Clustering methods, such as K-Means, DBScan, and Affinity Propagation, can be used to divide the solution space and direct the search to promising regions. These strategies can assist to broaden the search and prevent becoming locked in local optima.

While FCA has not been widely used for Knapsack issues, its potential benefits in this context are worth investigating. FCA offers a mathematical framework for conceptual clustering, which can help to find relevant subgroups of objects based on their properties. Using FCA-based clustering insights, it may be feasible to direct the search towards more promising parts of the solution space and enhance the effectiveness of optimization methods for Knapsack issues. Furthermore, FCA can improve the interpretability of solutions by showing underlying patterns and correlations among elements, allowing for better decision-making processes. As a result, combining FCA with GAs and clustering algorithms is a prospective route for improving the state-of-the-art in addressing Knapsack problems effectively.

Clustering algorithms have been used to improve search speed and solution quality when tackling Knapsack issues with metaheuristic techniques like GAs. However, while prior literature has mostly focused on GAs and clustering algorithms, there is still a need to investigate the possible benefits of incorporating FCA into the Knapsack problem optimization framework. This study intends to overcome this gap by exploiting FCA's capacity to extract meaningful ideas from data and uncover underlying structures, resulting in a complete technique for quickly addressing Knapsack issues.

3.1. Data Preparation

In this step, we create synthetic data to simulate instances of the Knapsack problem for testing and analysis. We have generated random values for item weights (w), item values (v), and the knapsack maximum weight capacity (W). The numpy package is utilized to generate these random numbers efficiently. First, we initialize two parameters: n, which is the number of items and can be adjusted to handle varying sizes of data, and R, the range of random values, which controls the variability and magnitude of the item values and weights. For example, setting $n = 100$ means that we will generate data for 100 items, and if $R = 1000$, the generated weights and values will be between 1 and 1000.

Next, we generate two lists of random integers using numpy's randint function. The list v contains the values of the items, each being a random integer between 1 and R, representing the worth or benefit of including an item in the knapsack. Similarly, the list w contains the weights of the items, also random integers between 1 and R, representing the cost or resource consumption of including an item in the knapsack. This random generation introduces variability, making the data more realistic for testing different scenarios of the knapsack problem.

Finally, the maximum weight capacity of the knapsack, W, is set to a percent of the total sum of all item weights. This ensures that the knapsack cannot hold all items, creating a need to select the optimal subset of items. The capacity is calculated by summing all weights and then scaling it down to a percent (which is adjustable) of the total, ensuring W is an integer value.

By adjusting n and R, we can experiment with different sizes and ranges of data, making the script flexible for various testing needs. Using numpy ensures the generation of random values is efficient and can handle large data sets, introducing necessary variability for realistic testing. Adjusting W as 75% of the total weight ensures that the problem is non-trivial, requiring a strategic selection of items to maximize value without exceeding the capacity.

4. Proposed Approaches

In this section, we'll look at multiple approaches to the Knapsack problem that combine Genetic Algorithms (GAs) with clustering techniques. Each part will look at a specific strategy, explaining how GAs are used with clustering approaches to improve solution quality and efficiency. In addition, we will present a novel technique, Genetic Algorithm with Formal Concept Analysis (FCA)-based Clustering, which uses FCA principles to refine the clustering process and improve optimization results.

To begin, the Genetic Algorithm (GA) will be introduced as the primary optimization strategy for solving the Knapsack issue. We will explain the fundamental concepts of GAs and their use in combinatorial optimization, emphasizing their strengths and limits in this context. The next subsections will look at how GAs may be integrated with various clustering techniques such as DBScan, K-Means, Affinity Propagation, and Mean-Shift. Each method will be explained, including how clustering contributes in dividing the solution space and directing the evolutionary search process to promising locations.

Among these methods, we will present a new approach: Formal Concept Analysis (FCA) based Clustering using Genetic Algorithm. FCA provides a mathematical framework for conceptual clustering, allowing for the identification of significant patterns within large datasets. By combining FCA with GAs, we want to use the data's intrinsic structure and relationships to influence the optimization process. Our technique aims to improve the clustering phase by exploiting FCA's interpretability and pattern recognition capabilities, resulting in higher solution quality and convergence while addressing the Knapsack problem.

4.1. Genetic Algorithm

Genetic algorithms are optimization approaches based on the process of natural selection in biological evolution. They are used to generate approximate solutions to optimization and search issues. The genetic algorithm in the given code attempts to solve the knapsack problem, a combinatorial optimization issue in which the goal is to maximize the overall value of objects placed in a knapsack while not surpassing its weight capacity.

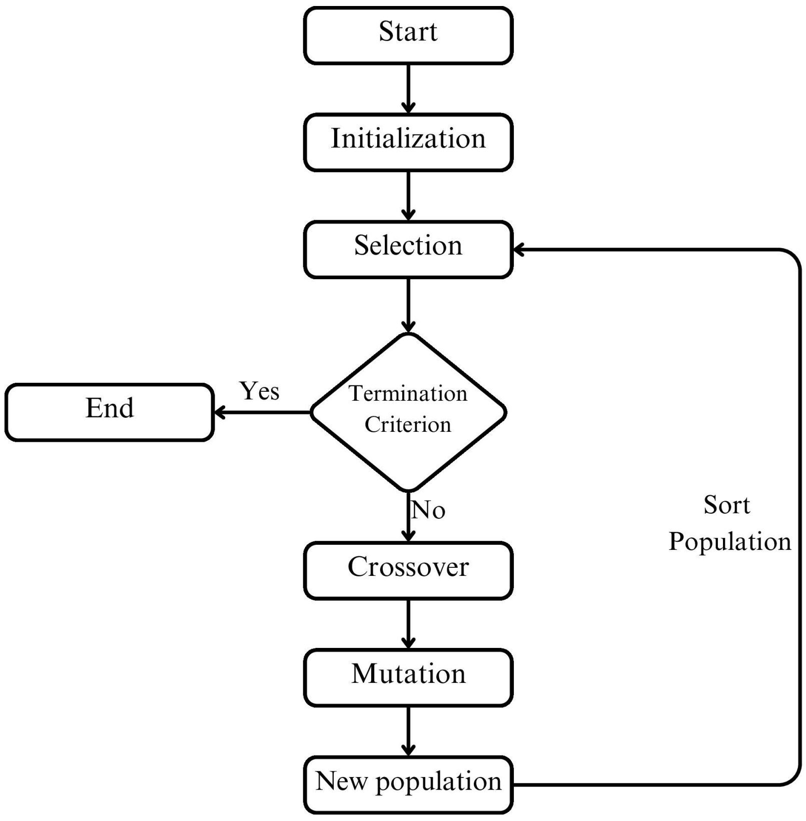

Figure 1. Flowchart of Genetic Algorithm.

The conventional genetic algorithm (GA) functions in a sequence of interrelated phases, as shown in figure 1. It all starts with the initialization step, in which a population of candidate solutions is generated at random. Each solution, frequently shown as a chromosome, is a possible solution to the problem at hand. Following initialization, the selection phase assesses the fitness of each solution, which is analogous to determining their capacity to live and reproduce in a population. Solutions with better fitness are more likely to be picked for subsequent steps.

Once the selection procedure is completed, the crossover (recombination) step begins. Pairs of chosen solutions exchange genetic information, similar to biological reproduction, resulting in new offspring. This interchange of genetic material enables for the investigation of various solution combinations, which may lead to better solutions. Subsequently, the mutation phase brings random modifications to the offspring solutions, allowing for the discovery of new solution spaces while still retaining genetic variety.

With the generation of children and the introduction of genetic diversity, a new population is generated by mixing the original parent solutions with the newly developed offspring. This new population goes through another phase of selection, with fitness re-evaluated and individuals picked for the following generation depending on their performance. This iterative procedure will continue until a termination requirement is fulfilled.

The termination phase determines if a predetermined stopping condition has been fulfilled. This condition might include completion of the maximum number of generations, achieving a particular level of fitness, or exceeding a computational time restriction. When the algorithm reaches the termination condition, it ends and returns the best solution obtained during the iterations. The flow of the classical GA captures the core of evolutionary processes, in which natural selection, genetic recombination, and random mutation collaborate to drive the exploration of solution spaces and optimize problem-solving results.

The genetic algorithm (GA) begins by generating a set of possible solutions. Each solution is represented by a binary vector x of length n. Each element $x_i \in \{0, 1\}$ specifies whether an item is included (1) or not included (0) in the knapsack. This random initialization gives a variety of beginning points for the program.

Next, the fitness of each candidate solution is assessed using the knapsack cost function. The fitness is calculated by adding the total value and weight of the chosen goods.

$$GainedValue = \sum_{i=1}^{n} v_i x_i \tag{4}$$

$$LostValue = \sum_{i=1}^{n} v_i (1 - x_i) \tag{5}$$

$$GainedWeight = \sum_{i=1}^{n} w_i x_i \tag{6}$$

$$LostWeight = \sum_{i=1}^{n} w_i (1 - x_i) \tag{7}$$

The gained value is calculated as equation (4), which is the dot product of the value vector v and the solution vector x. The value of items that are not included is represented by equation (5). Similarly, and represent the weight gained and lost are equation (6) and equation (7), respectively. If the total acquired weight exceeds the knapsack capacity W, the solution is punished by reducing the gained value to negative infinity, indicating an infeasible solution.

The selection procedure then employs a technique known as roulette wheel selection to identify parent solutions based on their fitness percentage. The cumulative probability for each solution is computed as equation (8).

$$c_i = \sum_{j=1}^{i} p_i \tag{8}$$

A random number r between 0 and 1 is created, and the solution for the lowest index i with $c_i > r$ is chosen. This means that solutions with higher fitness have a better chance of being chosen for reproduction.

During crossover (recombination), new offspring are formed by mixing components of two parent solutions. In single-point crossover, a random crossing point is selected, and the kids inherit the first portion of genes from one parent and the second part from the other, represented by equation (9) and equation (10).

$$y1 = x1[:index] + x2[index:] \tag{9}$$

$$y2 = x2[:index] + x1[index:] \tag{10}$$

Double-point crossover involves selecting two random crossover sites and inheriting segments between them from the opposite parent, denoted by equation (11) and equation (12).

$$y1 = x1[:index1] + x2[index1:index2] + x1[index2:] \tag{11}$$

$$y2 = x2[:index1] + x1[index1:index2] + x2[index2:] \tag{12}$$

Uniform crossover employs a binary mask (α) to determine gene inheritance, producing in equation (13) and equation (14),

$$y1 = \alpha \odot x1 + (1 - \alpha) \odot x2 \tag{13}$$

$$y1 = \alpha \odot x1 + (1 - \alpha) \odot x2 \tag{14}$$

where, $\odot$ denotes element-wise multiplication and α is a binary mask (vector) of the same length as the chromosomes, with each element being 0 or 1.

Mutation causes random changes to maintain genetic variety. Single-swap mutation swaps two neighboring elements, whereas double-swap mutation swaps two random elements, denoted by equation (15),

$$y[index[0]], y[index[1]] = y[index[1]], y[index[0]] \tag{15}$$

Uniform mutation flips a random gene, denoted by equation (16)

$$y[index] = 1 - x[index] \tag{16}$$

In inverse-swap mutation, a segment between two random locations is reversed, as represented by equation (17)

$$y = x[: index[1]] + reverse(x[index1: index2]) + x[index2:] \qquad (17)$$

The genetic algorithm iteratively executes selection, crossover, mutation, and replacement to create new generations. Each iteration assesses and picks the most effective solutions for the following generation. This process continues until a stopping requirement, such as a predetermined number of generations or convergence to a solution, is fulfilled. The objective is to develop the population toward optimal or near-optimal solutions to the knapsack problem.

4.2. Genetic Algorithm with DBScan Clustering

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering technique that uses point density to identify clusters, allowing for the discovery of clusters of any shape while dealing with noise. The approach relies on two parameters: ε (eps), the greatest distance between neighbors, and $MinPts$, the minimal number of points needed to form a dense area or cluster.

The program divides the data into three categories: core points, boundary points, and noise. A point (p) is called a core point if it has at least ε-neighborhood points $(MinPts)$. The mathematical definition of a point's ε-neighborhood is as follows in equation 18:

$$N_{\varepsilon}(p) = \{q \in D \mid dist(p, q) \leq \varepsilon\} \qquad (18)$$

$$dist(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \qquad (19)$$

Dist(p,q) is the distance between points p and q, commonly computed using the Euclidean distance equation (19).

Border points are those that are within the ε-neighborhood of a core point, whereas noise refers to points that are neither core nor border points. Clusters are generated by grouping core points and their related boundary points; noise points are disregarded throughout the clustering process.

Figure 2 depicts a hybrid genetic algorithm with DBSCAN clustering that combines classic genetic algorithm operations with clustering techniques to improve solution variety and quality. The procedure begins with the initialization step, in which a random set of potential solutions is produced. A preset cost function is then used to determine the fitness of each solution.

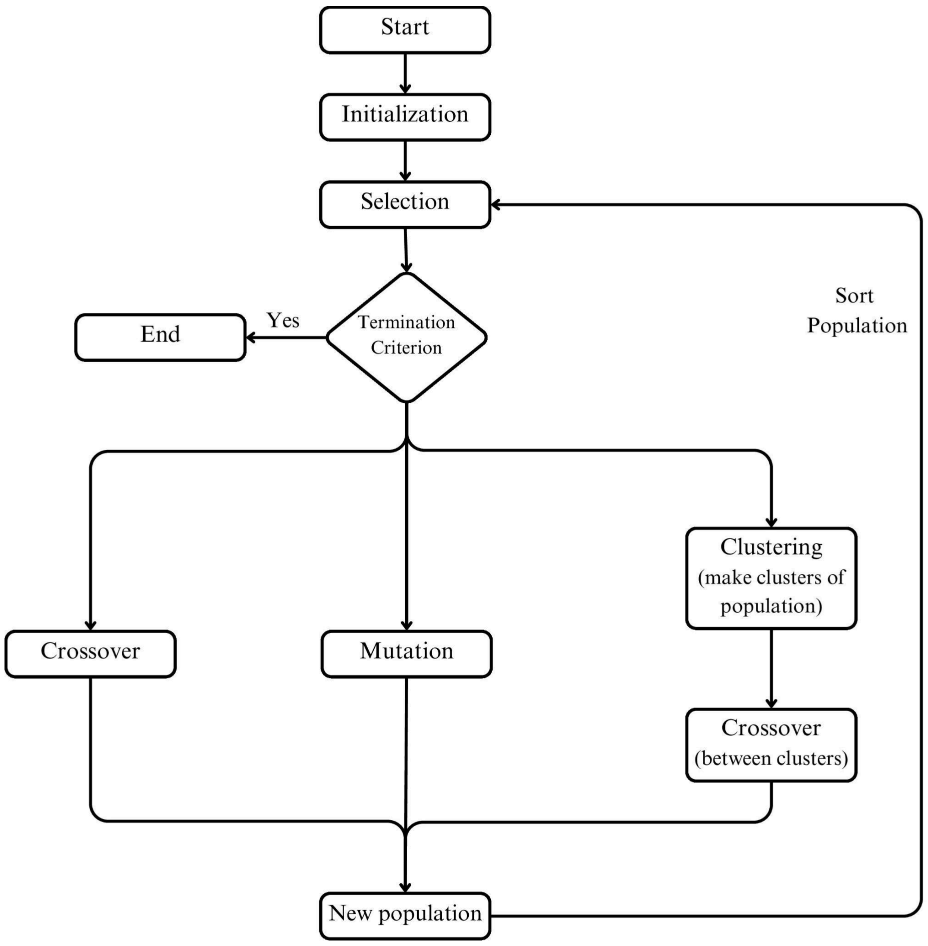

Figure 2. Flowchart of our proposed Genetic Algorithm with Clustering.

Following a fitness assessment, parent solutions are chosen depending on their fitness, generally utilizing methods such as roulette wheel selection. The next step is to execute standard genetic operations such as crossover and mutation. Crossover involves combining pairs of parents to generate children, whereas mutation brings random alterations to certain solutions in order to preserve genetic variety.

Next, DBSCAN clustering is used to group the entire population into clusters based on solution similarity. The Euclidean distance is used to determine distances between solutions, and clusters are created using the DBSCAN method. Inter-cluster crossover is then used to increase variety by picking one solution from one cluster and another from a different cluster, followed by the crossover.

The population is then updated by mixing newly generated solutions from standard and inter-cluster operations with the current population. A termination check determines if a stopping condition, such as a maximum number of generations or convergence requirements, has been reached. If not, the process is repeated from the fitness evaluation phase.

The hybrid genetic algorithm with DBSCAN clustering has two forms, both of which use cost functions, crossover, and mutation operations, as described in the preceding section on the genetic algorithm.

First Variation

In the first variant, the algorithm does the following steps:

- Initialization: Create an initial population and assess their fitness.
- Selection: Choose parent solutions depending on fitness.
- Standard Crossover and Mutation: Apply crossover and mutation to certain parents.
- DBSCAN Clustering: Use DBSCAN on the total population to detect clusters.
- Inter-Cluster Crossover: Choose one solution from one cluster and another from a separate cluster, then execute crossover to produce new offspring.
- Population Update: Add the kids to the population, assess their fitness, and choose the best options to create the new population for the future generation.

Second Variation.

The second variant deviates slightly:

- Initialization: Create an initial population and assess their fitness.
- Selection: Choose parent solutions depending on fitness.
- DBSCAN Clustering: Use DBSCAN on the total population to detect clusters.
- Inter-Cluster Crossover and Mutation: Combine solutions from several clusters. In addition, use mutation inside clusters by randomly picking solutions from clusters and adding modifications.
- Population Update: Incorporate the offspring and mutant solutions into the population, assess their fitness, and choose the best solutions to construct the new population for the following generation.

Distance computation is critical for the DBSCAN clustering process. The Euclidean distance is used to determine the similarity of solutions. Equation (19) calculates the Euclidean distance between two vectorized solutions (x and y). This distance metric is used to generate a distance matrix including the pairwise distances between all solutions in the population.

Cluster creation using DBSCAN consists of multiple processes. To begin, calculate the pairwise Euclidean distances between all solutions to create a distance matrix. The DBSCAN technique is then used to cluster the solutions based on their distance matrix. Core points are those having at least $MinPts$ neighbors within ε. Clusters are built by linking these core points with their border points. Clusters are tagged, and noise spots are found. Solutions within the same cluster are regarded as comparable, whilst those from separate clusters are considered varied.

Integrating DBSCAN clustering into the genetic algorithm offers numerous important benefits. One of the key advantages is greater diversity. By executing crossover between answers from distinct clusters, the method maintains a high genetic variety, preventing premature convergence and encouraging extensive exploration of the search space.

Furthermore, clustering allows the algorithm to explore diverse portions of the solution space more efficiently. By clustering similar answers together, it facilitates recombination between various solutions, which improves the search process. Another feature of DBSCAN is its tolerance to noise, which guarantees that outlier solutions do not have a harmful effect on the optimization process. Furthermore, the clustering stage dynamically adjusts to the distribution of solutions in the population, preserving variety while improving solution quality.

Overall, the hybrid genetic algorithm with DBSCAN clustering combines the advantages of genetic algorithms with density-based clustering. This combination improves optimization efficiency, making the hybrid technique especially useful for complicated search areas.

4.3. Genetic Algorithm with K-Means Clustering

The genetic algorithm (GA) with K-Means clustering is a sophisticated hybrid strategy that combines the advantages of genetic algorithms with the K-Means clustering technique to improve the optimization process. This strategy is especially beneficial for complicated issues where preserving variety and avoiding local optima is critical to finding high-quality solutions.

K-Means is a popular clustering technique that divides a dataset into k separate, non-overlapping groupings called clusters. K-Means' purpose is to minimize the within-cluster variance, which is the sum of the squared distances between each point and the cluster's centroid. The procedure begins by defining k random locations as centroids. It then iteratively assigns each point to the nearest centroid and updates the centroids depending on the average of the points given to each cluster.

The distance between a point (p) and a centroid (c_j) is determined using the Euclidean distance as in equation (19) where p is a point in the dataset and q is the centroid (c_j) of cluster j. Following the assignment stage, the centroid of each cluster is updated by averaging the points allocated to it (equation 20):

$$c_j = \frac{1}{|C_j|} \sum_{p \in C_j} p \tag{20}$$

where C_j is the set of points allocated to cluster j, and $|C_j|$ is the number of points in the cluster j. This procedure is repeated until convergence, at which point the assignments stop changing and the centroids stable. The end result is k clusters, each represented by a centroid. Each data point belongs to the cluster with the closest centroid to it.

The algorithm for the hybrid genetic algorithm with K-Means clustering is identical to that for the DBSCAN version (figure 2), with the main distinction being the clustering technique. Initially, a random set of candidate solutions is created, and their fitness is assessed using a predetermined cost function. Based on fitness, parent solutions are chosen using approaches similar to roulette wheel selection. Standard genetic operations, such as crossover and mutation, are then performed on these parents to produce offspring.

In the following stage, K-Means clustering is performed to the entire population to create clusters based on solution similarity. Inter-cluster crossover operations are carried out across solutions from various clusters to increase variety. The freshly developed solutions are merged with the current population, and the best solutions are chosen to form the new population for the following generation. This process continues until a stopping condition, such as a maximum number of generations or convergence criterion, is reached. The method's structure and phases were described in the preceding section on the hybrid genetic algorithm with DBSCAN clustering.

The K-Means clustering stage dynamically determines the number of clusters ($n_{clusters}$) based on the solution population. This is accomplished using a function that determines the maximum number of clusters permitted, which is a percentage of the total number of solutions and is determined by the $max\_clusters\_threshold$ argument. This guarantees that each cluster has an adequate number of solutions while also limiting the number of clusters to at least two to prevent trivial solutions. The calculation of the maximum number of clusters is given by equation (21):

$$max_{clusters} = max(2,\ int(len(solutions)\times max\_clusters\_threshold)) \quad (21)$$

Once the number of clusters has been identified, the K-Means technique is performed to the full dataset. The population is divided into k clusters, each with a centroid. Solutions are allocated to the nearest centroid based on Euclidean distance. Inter-cluster operations are then determined by the number of clusters established. If there is only one cluster, crossover occurs between randomly selected couples inside it. If there are numerous clusters, solutions from different clusters crossover to increase variety. This dynamic clustering technique divides the population into meaningful groups, allowing for efficient recombination while retaining genetic diversity.

Integrating K-Means clustering into the genetic algorithm provides numerous possible improvements over the traditional genetic algorithm. First, by executing crossover operations across solutions from distinct clusters, the algorithm maintains high genetic variety, limiting premature convergence and stimulating search space exploration. Second, the clustering procedure allows the algorithm to more effectively explore diverse portions of the solution space by grouping similar solutions together and promoting variation between clusters.

The algorithm's dynamic determination of $n_{clusters}$ adapts to population size and dispersion, ensuring scalability and effectiveness for complicated tasks. Overall, the hybrid genetic algorithm with K-Means clustering takes advantage of the strengths of both genetic algorithms and clustering techniques to improve optimization performance, making it ideal for complex problems where maintaining diversity and avoiding local optima is critical for finding high-quality solutions.

4.4. Genetic Algorithm with Affinity Propagation Clustering

Affinity Propagation (AP) is a clustering technique that finds exemplar points in the data and creates clusters based on them. Unlike classic clustering algorithms such as K-means or DBSCAN, which need a fixed number of clusters, AP finds the number of clusters based on the data. The procedure starts by generating a similarity matrix S. Each element $s(i,j)$ indicates the similarity between data points i and j. Typically, this similarity is measured as the negative squared Euclidean distance as in equation (22)

$$s(i,j) = -||x_i - x_j||^2 \quad (22)$$

AP works by exchanging messages between data points until a suitable collection of exemplars and clusters arise. There are two sorts of communications exchanged: responsibility $(r(i,j))$ and availability $(a(i,j))$. The obligation reflects point j's suitability as a model for point i, taking into account other possible options. It has been updated as equation (23):

$$r(i,j) \leftarrow s(i,j) - max_{j' \neq j}\{a(i,j') + s(i,j')\} \quad (23)$$

Availability assesses the suitability of point i to pick point j as an example, taking into account the support from other points. It's updated as equation (24):

$$a(i,j) \leftarrow min(0, r(j,j) + \sum_{i' \notin \{i,j\}} max(0, r(i',j))) \quad (24)$$

The algorithm iteratively changes these messages until they reach convergence. The preference parameter p, usually set to the median similarity, affects the number of clusters. Higher values of p lead to fewer clusters, while lower values lead to more.

The integration of AP with GA is comparable to the previously described GA with K-means or DBSCAN clustering. The major distinction occurs during the clustering process, in which AP is used to group the population. The algorithm starts with an initial population of solutions and assesses the fitness of each one. AP is then used to cluster the population based on commonalities. Within each cluster, crossover and mutation processes are used to produce new progeny. The best individuals are then chosen to create the new population, and the process is repeated for a set number of iterations or until the convergence requirements are reached. The precise processes and flow were fully covered in earlier sections, which discussed the GA's overall structure in conjunction with clustering approaches (figure 2).

Genetic Algorithm with Affinity Propagation Clustering is implemented in two versions: run_AP1 and run_AP2. Both functions have a similar structure as those mentioned in earlier parts (4.2, 4.3), with slight differences in implementation details. Initially, the starting population ($Ipop$) and their associated costs ($Icosts$) are duplicated, and several lists are created to hold the best solutions, their costs, execution durations, and the number of function evaluations (NFE).

Within the main loop, the population is subjected to crossover, mutation, and selection processes for a predetermined number of iterations ($maxIt$). AP Clustering is used to group the population. The method separates solutions into clusters based on their similarity, then crossover operations are done inside each cluster to produce offspring. Mutation procedures are used to further diversity the population. The offspring solutions are assessed for fitness, and the combined pool of parents and offspring is sorted, with the best individuals chosen to create the new population. The best options and their prices are recorded at each iteration, and the findings are saved to a file for further study.

The use of AP Clustering into GA has numerous possible benefits over a traditional GA. Unlike K-means and other clustering algorithms, which need a fixed number of groups, AP identifies the correct number of clusters depending on the data, resulting in more natural grouping of solutions. By grouping the population, the algorithm can concentrate on crossover and mutation operations inside and across clusters, encouraging diversified exploration of the solution space and perhaps avoiding local optima. The clustering mechanism adjusts dynamically to the population structure in each iteration, allowing the algorithm to respond to changes in the population's distribution while maintaining a balance between exploration and exploitation. AP can easily handle huge populations and complicated similarity structures, making it suited for issues with significant population diversity. Overall, the Genetic Algorithm with Affinity Propagation Clustering combines the qualities of both approaches to possibly enhance convergence rates and solution quality, making it a viable alternative to classic genetic algorithms.

4.5. Genetic Algorithm with Mean-Shift Clustering

Mean Shift(MS) is a clustering approach that does not need previous knowledge of the number of clusters. Using a kernel density estimation approach, it iteratively moves data points towards the mode (the maximum density of data points). The process begins by picking a kernel function, often a Gaussian kernel, and applying it to each data point. A Gaussian kernel is a mathematical function that provides weights to surrounding locations based on their distance from a certain point.The kernel's effect is computed as equation (25):

$$K(x) = exp(-\frac{||x-x_i||^2}{2h^2}) \tag{25}$$

In this equation (25), x is a point in the dataset, x_i is another point, and h is the bandwidth parameter that determines the kernel width. The Mean Shift vector $m(x)$ for each point x is calculated by taking the weighted average of all points in the dataset as shown in equation (26).

$$m(x) = \frac{\sum_i K(x-x_i)x_i}{\sum_i K(x-x_i)} \tag{26}$$

The method transfers each point x to the position of its Mean Shift vector until convergence, which occurs when the shift is less than a predetermined threshold. After convergence, points that converge to the same mode are combined into a single cluster.

The integration of MS Clustering with a GA is comparable to integrating with other clustering algorithms like DBSCAN, K-means, and AP. The primary distinction is the clustering stage, in which MS is utilized to group the population. The algorithm starts with an initial population of solutions and assesses the fitness of each one. MS is then used to cluster the population according to the density of solutions. Within each cluster, crossover and mutation processes are used to produce new progeny. The best individuals are chosen to create the new population, and the process is repeated for a set number of iterations or until the convergence requirements are reached. The exact processes and flow were fully outlined in the preceding sections, which covered the overall framework of the GA coupled with clustering approaches (figure 2).

The *run_MS1* and *run_MS2* functions implement the GA with MS Clustering in two versions.Both functions have a similar structure as those mentioned in earlier parts (4.2, 4.3, 4.4). Initially, the starting population ($Ipop$) and their associated costs ($Icosts$) are duplicated, and several lists are created to hold the best solutions, their costs, execution durations, and the number of function evaluations (NFE).

In the main loop, the population is selected, crossovered, and mutated for a set number of iterations ($maxIt$). Mean Shift Clustering is used to cluster the population. The method determines clusters based on the density of solutions, and crossover operations are done inside each cluster to produce offspring. Mutation procedures are used to further diversity the population. The offspring solutions are assessed for fitness, and the combined pool of parents and offspring is sorted, with the best individuals chosen to create the new population. The best options and their prices are recorded at each iteration, and the findings are saved to a file for further study.

The integration of Mean Shift Clustering into the Genetic Algorithm has numerous possible benefits over a traditional GA. Unlike K-means and other clustering algorithms, which need a fixed number of clusters, MS calculates the number of clusters depending on the data distribution. By grouping the population, the algorithm can concentrate on crossover and mutation operations inside and across clusters, encouraging diversified exploration of the solution space and perhaps avoiding local optima. The clustering mechanism adjusts dynamically to the population structure in each iteration, allowing the algorithm to respond to changes in the population's distribution while maintaining a balance between exploration and exploitation. MS can easily handle complicated density patterns and non-linear correlations in data, making it appropriate for situations involving a diverse population. Overall, the GA with MS Clustering combines the qualities of both approaches to possibly enhance convergence rates and solution quality, making it a strong alternative to regular GAs.

4.6. Genetic Algorithm with Formal Concept Analysis based Clustering

Formal Concept Analysis (FCA) is a mathematical framework that extracts and analyzes links between objects and their qualities, resulting in a structure known as a concept lattice. This method provides a formal framework for data analysis and grouping. The main components of FCA are:

- Formal Context: A formal context is represented as a triple $K = (G, M, I)$, where G is a set of objects, M is a set of attributes, and $I \subseteq G \times M$ is a binary relation showing which objects have which attributes. For example, if $(g, m) \in I$, then object g has attribute m.

- Formal Concepts: A formal idea is a pair (A, B) where $A \subseteq G$ (the extent) is the set of things that share common qualities, and $B \subseteq M$ (the intent) is the set of attributes shared by those objects. These satisfy equations 27 and 28:

$$A = \{g \in G \mid \forall m \in B,\ (g, m) \in I\} \tag{27}$$

$$B = \{m \in M \mid \forall g \in A,\ (g, m) \in I\} \tag{28}$$

  This indicates that the extent consists of all objects that share all characteristics with the purpose, and the intent consists of all attributes shared by all objects in the extent.

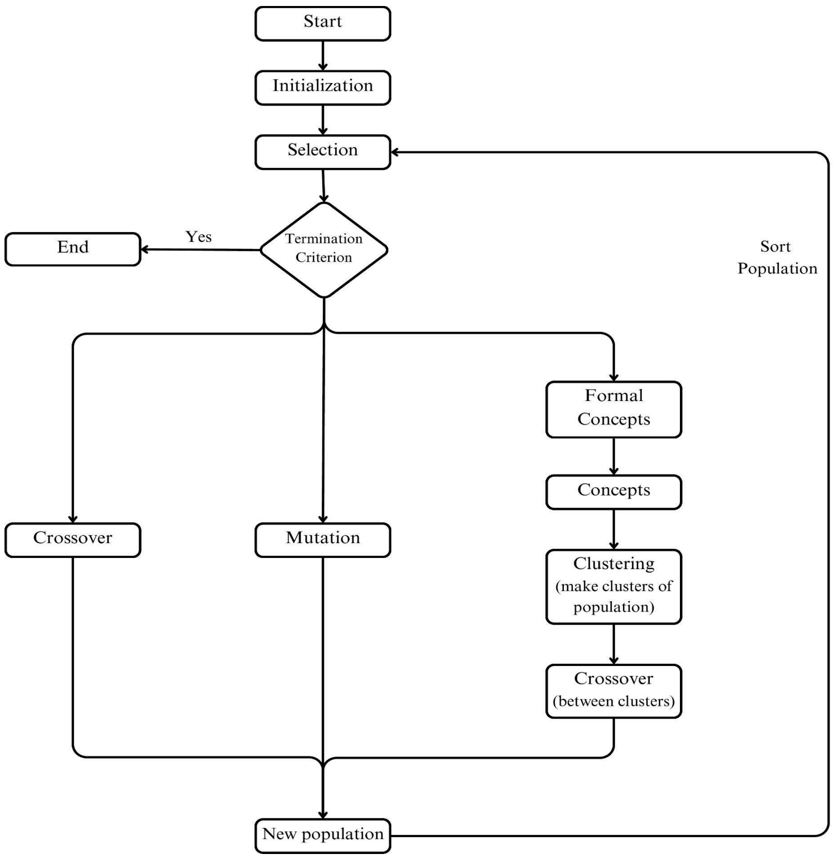

Figure 3. Flowchart of our proposed Genetic Algorithm with Formal Concept Analysis based Clustering.

The evolutionary algorithm augmented with FCA-based clustering follows these phases, as indicated in Figure 3:

- Start: The algorithm begins.
- Initialization: Generate a random sample of alternative solutions.
- Selection: Choose the best solutions from the population depending on their fitness levels.
- Termination Check: Determine if the termination requirements are fulfilled, such as completing the maximum number of iterations or attaining a satisfying

solution. If the conditions are satisfied, the algorithm terminates; otherwise, it continues.

- Crossover: Use a crossover operation on 70% of the population to produce new offspring.
- Mutation: Use a mutation operation on 10% of the population to induce diversity.
- Create a formal context with the entire people.
- Concept Generation Using the Sofia method: Create ideas with the Sofia method, which is efficient for huge datasets.
- Clustering: Create clusters of solutions based on the provided concepts.
- Crossover Between Clusters: Use crossover operations between clusters to produce new solutions.
- New Population Formation: Create a new population using the best solutions available.

The FCA clustering process begins with the *perform_concept_clustering* function, which transforms the input population to binary data format. This transformation makes use of one-hot encoding, a technique that turns categorical data into a binary matrix, guaranteeing that each potential value of a categorical characteristic is represented as a distinct binary feature. This phase is critical for establishing a formal context since FCA works with binary data.

Once the data is transformed, a formal context is established. The formal context, represented as $K = (G, M, I)$, includes a collection of objects (G), a set of attributes (M) , and a binary relation $(I \subseteq G \times M)$ that indicates which objects have which attributes. The formal context serves as the basis for creating a concept lattice that includes all conceivable groups of objects and their related properties.

The Sofia method is used to build the ideal lattice, which is well-known for its efficiency when dealing with big data sets. The Sofia algorithm creates the lattice slowly, adding items one at a time and changing the structure as needed. For each new item, it arranges the sets of objects (extents) and their accompanying properties (intents) hierarchically. This technique yields a complete lattice that depicts all relevant groups of objects based on their properties.

The Sofia algorithm is fundamental to the FCA-based clustering process. It creates the ideal lattice iteratively, beginning with an empty lattice and adding things successively. The method updates the lattice with each new item by recognizing pairings (A,B) of objects and characteristics. These pairs create the lattice's nodes, which are hierarchically structured to reflect the inclusion connections between groups of objects and characteristics.

The Sofia algorithm is efficient because it uses incremental updates, which eliminates the need to construct the whole lattice from scratch for each new object. This method is especially useful for huge datasets since it minimizes computational complexity while ensuring scalability. By keeping an up-to-date lattice structure, the Sofia method makes it easier to generate and enhance clusters based on formal notions.

After generating the lattice, ideas are filtered to exclude trivial or too big groups. Concepts with extents equal to the number of attributes, extents with fewer than two objects, and extents that encompass more than 40% of the total objects are all removed. This filtering guarantees that only meaningful and manageable clusters are saved for later processing.

The *refine_clusters* function cleans up the clusters by eliminating duplicates and guaranteeing that each cluster includes unique items. This refining process guarantees that the final clusters are relevant and useful for future genetic operations. By removing duplicate and excessively tiny clusters, the method focuses on major groupings that might aid in the optimization process.

The genetic method is implemented using FCA-based clustering, which combines genetic processes with formal concept analysis to improve the optimization process. An explanation of the code is provided below:

- Initialization: The starting population and related expenditures are established. The population comprises possible solutions to the optimization problem, each represented as a binary string or array. The expenses are evaluated using a fitness function that assesses the quality of each option.
- Main Loop: The algorithm iterates for a certain number of times (maxIt). In each cycle, genetic processes and FCA-based clustering are used to improve the population's solutions.
    - Crossover and Mutation: As described in Section 4.1, the crossover operation mixes pairs of solutions to generate new offspring, therefore increasing genetic variety. Crossover involves selecting pairs of solutions (parents) and swapping segments of their binary strings to generate offspring. The mutation procedure adds minor random modifications to certain solutions by flipping bits in binary strings, preserving population diversity.
    - FCA and Clustering: The function perform_concept_clustering uses FCA to construct clusters from the current population. The function translates the population to binary format, establishes a formal context, applies the Sofia algorithm to construct a concept lattice, and then filters the ideas to keep important clusters.
    - Cluster-Based Crossover: Crossover activities occur inside and between clusters. Solutions from several clusters are joined to form new solutions, ensuring a broad investigation of the solution space. This stage uses the relevant groups generated by FCA to guide genetic procedures.
- Selection and Update: The best solutions from the new population are chosen based on fitness. This selection procedure guarantees that only the most promising options move on to the next iteration. The population is sorted by fitness, and the best solutions are kept, producing the population for the following iteration.

- Evaluation and Logging: Throughout the iterations, the algorithm records the best solutions, their costs, and the time spent on each iteration. This logging offers information on the algorithm's performance and convergence characteristics.

The *run_FCA1* and *run_FCA2* functions implement specialized techniques for using genetic operations alongside FCA-based clustering. These routines manage the genetic algorithm's main loop, which includes crossover, mutation, and clustering operations, as well as repeated population updates.

The *run_FCA1* function favors crossover inside the FCA-identified clusters, whereas the *run_FCA2* method balances crossover and mutation within and across clusters. Both functions employ the *perform_concept_clustering* function to create clusters and the *refine_clusters* function to refine them. The key distinction is the emphasis on various genomic procedures and their implementation within the defined clusters.

Integrating Formal Concept Analysis (FCA) with genetic algorithms provides several benefits over regular genetic algorithms. To begin, FCA-based clustering increases the variety of the solution space. By grouping solutions based on similar qualities, the algorithm boosts genetic variety and reduces the danger of early convergence to local optima. This diversity is essential for a thorough investigation of the solution space, improving the possibility of finding optimum solutions.

Furthermore, the inclusion of FCA allows for better investigation of the solution space. By focusing crossover operations inside and across clusters, the algorithm may efficiently travel specified portions of the solution space. Clusters, which indicate significant groupings of solutions, allow for targeted exploration, improving the algorithm's capacity to find high-quality answers rapidly.

FCA-based clustering generates meaningful groups, which adds to the approach's benefits. Genetic operations become more focused and meaningful when answers are grouped according to shared traits. The ideal lattice, which defines connections between objects and qualities, directs genetic operations toward combinations that are likely to provide superior results. This organized strategy increases the efficacy of genetic operations, resulting in more efficient optimization procedures.

Furthermore, the approach's scalability stands out. Taking use of the Sofia algorithm's efficiency in processing huge datasets, the integrated method is scalable even for complicated optimization issues. The gradual creation of the idea lattice reduces computational cost, allowing the method to handle and analyze enormous populations with minimal overhead. This scalability is especially useful for real-world applications that deal with large datasets.

The combination of FCA and genetic algorithms promotes better convergence to high-quality solutions. By utilizing relevant clusters, the algorithm may reach optimum solutions faster. Targeted genetic operations inside clusters allow for comprehensive study of interesting portions of the solution space, increasing the total convergence rate. This rapid convergence minimizes the computing work needed to achieve adequate results, making the technique more economical.

Finally, the genetic algorithm with FCA-based clustering combines the capabilities of both approaches to provide an effective tool for handling difficult optimization issues. This integrated method surpasses standard genetic algorithms in terms of diversity, exploration, and convergence by increasing variety, allowing focused exploration, encouraging meaningful groupings, assuring scalability, and boosting convergence.

5. Experimental Results

In this section, we discuss the findings from our study to assess the performance of all methods for solving combinatorial optimization problems, with a special emphasis on the Knapsack problem. This section is divided into two main sections. In section 5.1, we specify our process of fine-tuning the parameters, we explain the challenges we faced and the methods we used to solve them, and finally, we arrive at the optimized parameter set that we used in our implementations. Section 5.2 provides a detailed analysis of our experimental results with four different benchmark datasets. In particular, all computational processes were carried out using the computational resources of HSE University's high-performance computing (HPC) facilities, highlighting the indispensable assistance provided by these resources as recognized in Kostenetskiy, Chulkevich, and Kozyrev's work (2021).

5.1. Calibration analysis

When optimizing combinatorial problems using Genetic methods (GA) and clustering methods, parameter calibration is crucial in getting optimal performance. This section digs into the calibration analysis of numerous parameters that were examined, their effect on the results, and the reasoning behind the final numbers.

The primary parameters influencing the performance of GA and clustering algorithms are:

- nPop (Population Size): The number of solutions in a single population. Larger populations broaden the search space but increase computing effort. The tested values were 100 and 200.
- nc (Crossover Rate) refers to the chance of chromosomal pairs crossing over. A greater rate promotes variety but may disrupt high-quality solutions. The selected value is 0.7.
- nm (Mutation Rate) refers to the chance of a chromosomal mutation. Mutation creates variety, preventing premature convergence. The selected value is 0.2.
- R (Range) refers to the maximum value or weight an item can have.
- W (Maximum Weight): The knapsack's weight limit, tested as a fixed (5000) and dynamic (75% of total item weight). The dynamic value of 0.75 was selected.
- maxIt (Maximum Iteration): The number of iterations in a single run. More iterations improve solution quality while increasing computing time. The chosen value is 2000.
- Cluster Rate (nclu): The percentage of the population used for clustering. The tested numbers were 20% and 40%.

During the experimental phase, numerous parameter combinations were examined to determine the best values for various algorithms. Initial testing with lower populations and fewer iterations produced unsatisfactory convergence and solution quality. Increasing the population size from 100 to 200 greatly increased the diversity of solutions, but at a larger computational cost. A crossover rate of 0.7 achieved a compromise between preserving variety and not disturbing effective solutions too frequently. A mutation rate of 0.2 proved successful in introducing sufficient variants while avoiding excessive unpredictability. Setting the maximum iterations to 2000 allowed for enough exploration of the search space, which was crucial for obtaining near-optimal answers.

For clustering-based genetic algorithms, a cluster rate of 0.2 was chosen since it offered enough clustering influence without overpowering the natural selection process. DBSCAN, K-Means, Affinity Propagation, Mean-Shift, and FCA each required precise parameter values. For example, DBSCAN1 with a cluster rate of 0.2 and min_samples set to 2 achieved successful noise management and cluster formation. KMEANS1 with a cluster rate of 0.2 and min_clusters set to 2 performed well in initial centroid placements, considerably improving convergence speed. Affinity Propagation (AP1) with a cluster rate of 0.2, damping set to 0.9, and convergence_iter set to 15 achieved a compromise between stability and convergence speed.

The optimal parameters identified throughout these experiments are shown in Table1. These parameters consistently produced the highest possible results.

GA and Other Algorithms	**DBSCAN1**	**KMEANS1**	**AP1**	**FCA1**	**MS1**
nc = 0.7 nm = 0.2 R = 1000 W = 0.75 nPop = 200 maxIt = 2000	min_samples=2 nclu = 0.2	min_clusters = 2 nclu = 0.2	damping = 0.9 convergence_iter = 15 nclu = 0.2	algo= ‘sofia’ L_max = 100 max_object = 0.4 nclu = 0.2	quantile = 0.2 n_samples = 500 min_bandwidth = 0.1 nclu = 0.2

Table 1. Optimal Parameters.

A crossover rate of 0.7 was chosen to retain genetic diversity while preserving the quality of fit solutions. A mutation rate of 0.2 was chosen to introduce enough variety without causing significant disruption. The number of runs was set to 5 to allow a thorough statistical examination. Using a dynamic weight value (75%) improves adaptability to the problem's particular restrictions. A greater population size of 200 provides varied answers but necessitates more processing. The maximum number of iterations was set at 2000 to allow the algorithm to converge to an optimal solution. To balance influence and genetic processes, cluster rates of 0.2 were shown to be best across clustering methods. By calibrating these parameters, the ideal configurations were discovered, improving GA and its clustering variants' efficacy in tackling complicated combinatorial optimization problems.

5.2. Comparison of algorithms and discussion

Each clustering approach has varied performance characteristics across various sample sizes (100, 300, 500, and 800). The Genetic Algorithm (GA) as shown in Table 2, regularly exhibits steady and efficient performance, with low variability and minimal CPU time, making it a dependable option for combinatorial optimization tasks.

- **Small Sample Size (n_sample = 100)**

  In a scenario of small sample size, the Genetic Algorithm (GA) produced consistent results with low variability (std = 15.16) and minimal CPU time (135.45 seconds). This demonstrates that GA is efficient and stable for smaller datasets, resulting in consistent outcomes with no processing overhead.

  DBSCAN1 and DBSCAN2 had increased variability and much longer CPU times, indicating that while these approaches may uncover various solutions, they are computationally costly for smaller datasets. The larger interquartile ranges and the appearance of outliers in their box plots suggest that these algorithms investigate a bigger variety of solutions, albeit at the expense of efficiency and stability.

  KMEANS1 and KMEANS2 have similar variability to GA, but needed more computing resources. The somewhat increased CPU timings for KMEANS versions reflect the added computational work required for clustering procedures.

Nonetheless, their solution quality was competitive, giving them feasible options when computing resources aren't a top priority. AP1 and AP2, as well as FCA1 and FCA2, displayed intermediate performance in terms of balancing variability and CPU time. The box plots reveal that these algorithms exhibit moderate variability, indicating a balanced approach to exploration and exploitation. However, AP2 and FCA2 were less efficient than their peers, underlining the importance of parameter adjustment to improve their performance on smaller datasets.

MS1 and MS2 demonstrated significant variability, with MS2 being particularly computationally demanding. The increased CPU time and the occurrence of outliers in their box plots indicate that Mean Shift clustering, particularly MS2, may not be appropriate for smaller datasets due to its greater computing needs and less steady performance. MS1 had a little improved balance, showing possibility for future tuning.

- **Medium Sample Size (n_sample = 300)**

  For medium sample sizes, GA remained stable with low variability (std = 114.02) and a moderate CPU time (334.04 seconds). This consistency demonstrates GA's durability across a variety of dataset sizes, delivering consistent performance with little oscillations.

  DBSCAN1 and DBSCAN2 continued to exhibit increased variability and CPU times, showing that their resilience is at the expense of efficiency. The line plots illustrate that, while DBSCAN variations may produce high-quality results, their convergence is slower and they need more processing resources, rendering them unsuitable for medium-sized datasets.

  KMEANS1 and KMEANS2 functioned well, although KMEANS2 exhibited more variability. The box plots show that, despite its larger variability, KMEANS2 may identify better solutions with more iterations, indicating its potential for more comprehensive exploration of the solution space. However, the higher CPU time for KMEANS2 is a trade-off that necessitates additional processing resources.

AP1 and AP2 performed rather well, with AP1 being more efficient than AP2. The line plots demonstrate that AP1 converges rapidly, comparable to GA, but AP2 takes longer and has more fluctuations, suggesting that AP1 is better suited to medium-sized datasets where efficiency is critical.

FCA1 and FCA2 showed greater unpredictability than GA, although with reasonable CPU times. The box plots show that, while these algorithms may explore a wide range of solutions, their performance is less consistent than GA. However, FCA1's CPU time remained competitive, giving it a feasible option when some fluctuation is tolerated.

MS1 performed well again, with minimal variability and a fair CPU time, however MS2 was less efficient due to the high CPU time needs. The line graphs show that MS1 performs consistently and efficiently, but MS2's greater processing needs render it unsuitable for medium-sized datasets.

- **Large Sample Sizes (n_sample = 500, n_sample = 800)**

  With large sample sizes, GA maintained its reliability, with low variability and competitive CPU speeds (456.27 seconds for 500 samples and 707.40 seconds for 800 samples). GA's constant performance across all sample sizes demonstrates its durability and effectiveness in managing huge datasets.

  DBSCAN1 and DBSCAN2 had much greater variability in CPU times, indicating scalability problems. The box and line graphs show that, while these algorithms may identify a variety of solutions, their computing requirements make them unsuitable for bigger datasets.

  KMEANS1 and KMEANS2 demonstrated modest variability, although KMEANS2 needed significant computing resources. The increased CPU time for KMEANS2 demonstrates the trade-off between solution quality and computational effort.

  AP1 and AP2, particularly AP1, kept variability under control while maintaining respectable CPU times. The box plots show that AP1's performance is consistent, making it ideal for big datasets where efficiency is critical.

  FCA1 and FCA2 exhibited greater fluctuation, while FCA1's CPU time remained comparable. The line graphs indicate that FCA1 can discover high-quality solutions with little computational effort, however FCA2's increased variability and CPU times render it unsuitable for big datasets.

MS1 had great performance with low variability and moderate CPU time, but MS2 displayed excessive variability and CPU times, exposing possible inefficiencies in certain cases. The box plots show that MS1 is a good competitor for situations needing stability and efficiency in cost maximization, but MS2's greater computing needs render it unsuitable for bigger datasets

The graphical representations (Figure 5 and Figure 6) below provide visual insights into the variability and distribution of algorithm performance across numerous runs. Figure 5 adds to our knowledge by displaying how each algorithm's cost values change over iterations.

Figure 5 depicts a convincing visual depiction of the performance distribution for each algorithm over numerous runs, allowing us to comprehend not only the average performance but also the variability and dependability of each approach under various situations. This helps to understand the variability, consistency, and relative optimization efficacy of Genetic Algorithms and Clustering-Enhanced Genetic Algorithms.

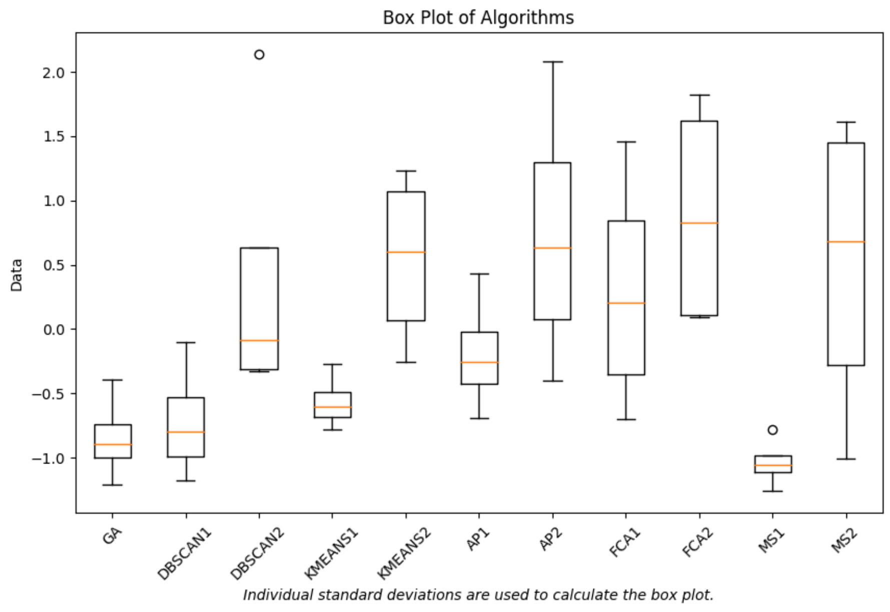

Figure 5. Box Plot of Standard Deviation of Costs of all Algorithms.

	Test 1 (n_samples = 100)					**Test 2 (n_samples = 300)**					**Test 3 (n_samples = 500)**					**Test 4 (n_samples = 800)**				
	Max	**Min**	**Avg**	**Std**	**CPU**	**Max**	**Min**	**Avg**	**Std**	**CPU**	**Max**	**Min**	**Avg**	**Std**	**CPU**	**Max**	**Min**	**Avg**	**Std**	**CPU**
GA	**46118**	46081	46092.4	**15.16**	**135.45**	148079	147797	147992.2	114.02	**334.04**	**249426**	248967	249173.2	**182.31**	**456.276894**	375143	374716	374895.2	**199.09**	**707.40**
DBSCAN1	**46118**	46037	46082.4	38.25	5935.96	**148114**	147977	148056	**50.24**	12223.96	249231	248716	248962.6	182.46	21692.32035	374924	374014	374558.6	386.18	28751.87
DBSCAN2	**46118**	46016	46080.8	43.19	6621.32	146923	146273	146570.4	318.39	11927.45	237625	236729	237155.2	341.47	21928.50208	339926	338001	338776.8	724.34	26408.21
KMEANS1	**46118**	46036	46059.2	34.680	2611.92	147918	147682	147821.2	100.14	4052.93	248575	248014	248368.8	219.91	3542.577268	373623	372515	373057.4	409.39	4486.54
KMEANS2	46081	45921	46033.4	65.92	3643.46	147667	147033	147340.4	228.07	12485.25	239388	238327	238930.6	465.16	4096.822262	340552	338623	339815.4	798.33	5291.33
AP1	**46118**	46036	46073	33.42	250.44	147840	147653	147736.4	89.47	508.55	248244	247326	247842.4	376.26	748.976232	366210	362240	364566.8	1485.81	985.15
AP2	**46118**	46043	46075.4	32.11	209.25	148013	147579	147839.2	164.75	810.44	246845	245430	246175.8	683.78	546.08536	359304	351134	356153.2	3131.72	596.00
FCA1	**46118**	45921	46024.2	70.76	325.68	147903	147426	147670.4	197.23	740.61	248457	247616	248164	360.74	1283.949852	374030	373176	373427.4	352.81	1942.72
FCA2	46061	45873	45997.4	72.57	233.63	146906	146508	146758.8	153.19	2414.98	239820	237980	238921	884.57	10910.62117	340747	338307	339412.6	1161.39	19428.93
MS1	**46118**	46081	46097.8	18.55	5041.03	**148115**	147996	148058.8	**43.87**	8442.07	**249415**	249079	249229	**148.95**	11450.84953	**375279**	374554	374857	272.15	12386.97
MS2	46091	46044	46077.6	19.434	6436.59	147352	146967	147162.2	142.93	6850.62	241754	239955	240990.8	775.35	11715.7507	352535	345751	350205	2659.93	16415.15

Table 2. Costs of all Tests.

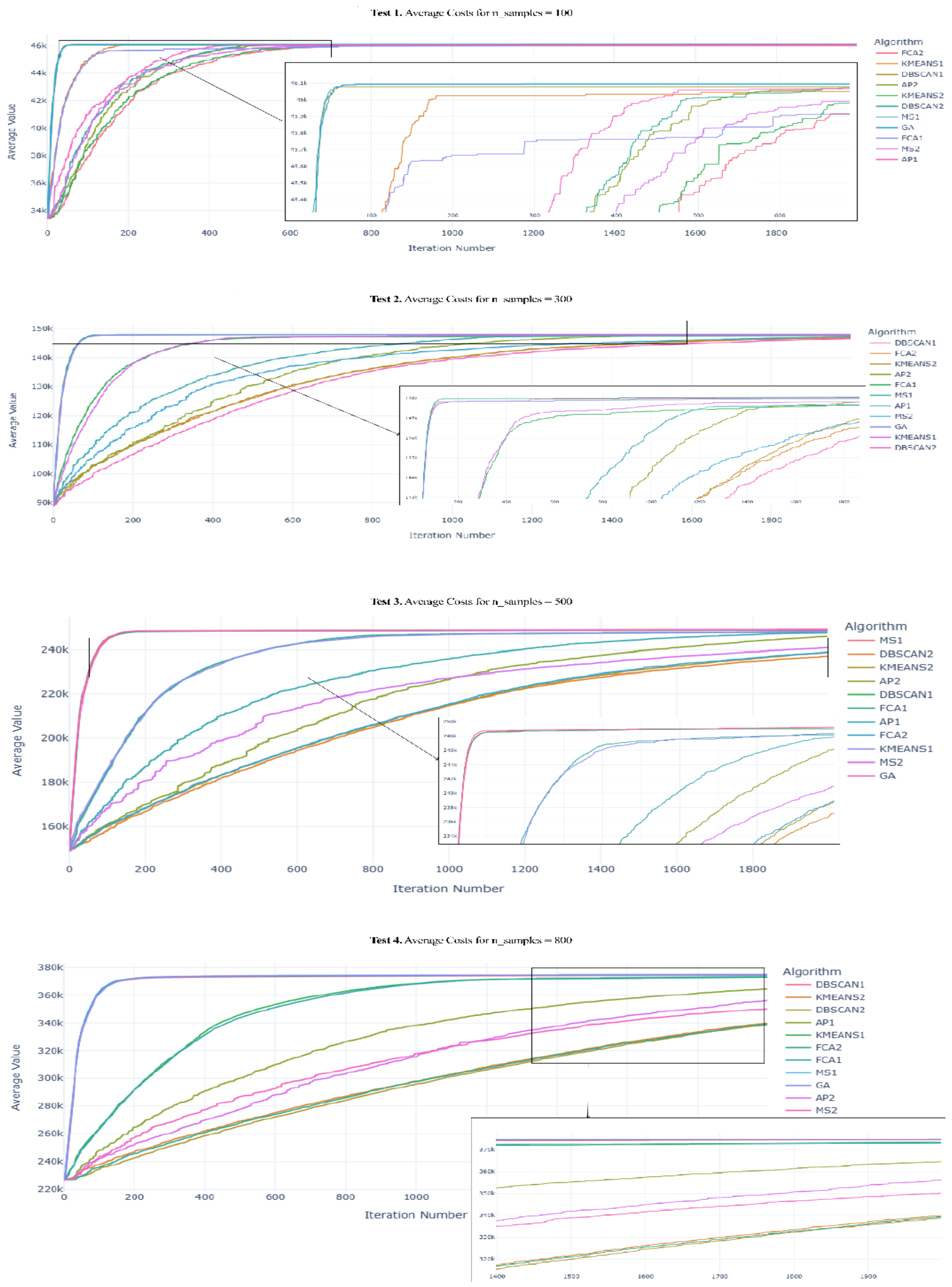

Figure 6. Plot of costs of all Algorithms in each Test.

Figure 6 shows the convergence tendencies and optimization efficiency of several clustering-based genetic algorithms across iterations, as compared to the Genetic Algorithm. This research takes a close look at how each algorithm improves in terms of cost reduction across various sample sizes. Each line plot shows the average costs for algorithms across several iterations, providing information on the speed of convergence, the efficacy of cost maximization, and the stability of each algorithm's performance over time.

- **Genetic algorithm (GA)**

  The box plot (figure 5) for the Genetic Algorithm (GA) shows a narrow distribution of cost values, indicating good stability in performance over several iterations. The interquartile range (IQR) is narrow, indicating that the majority of data points are near to the median, with few outliers. This demonstrates that GA may give consistent results with minimal variability, making it a viable alternative for combinatorial optimization jobs. The low variation in GA results is critical for cost maximization issues since it assures that the algorithm continually generates near-optimal solutions with minimal deviations.

  The line plot (figure 6) for GA shows a consistent and smooth convergence trend across iterations. The cost values continuously increase, demonstrating the algorithm's capacity to effectively search and utilize the solution space. The line plot shows that GA swiftly discovers good solutions early in the process and refines them over future rounds. This gradual progress without abrupt oscillations demonstrates GA's ability to carefully navigate the solution space, ensuring that solutions converge to optimal values in a predictable way.

  When compared to other algorithms, GA consistently demonstrates reduced variability and faster convergence. This makes it ideal for applications requiring high solution stability and computational efficiency. Unlike clustering-based algorithms, which may display increased variability owing to their investigation of multiple solution spaces, GA maintains a concentrated search while efficiently balancing exploration and exploitation. This balance guarantees that GA avoids the hazards of early convergence while also avoiding becoming caught in local optima, making it an excellent candidate for a variety of optimization tasks.

- **DBSCAN Variants: (DBSCAN1, DBSCAN2)**

  The box plots (figure 5) for DBSCAN1 and DBSCAN2 reveal substantially more variability than GA, with larger interquartile ranges and more outliers. This implies that, while DBSCAN may identify a variety of answers, its findings are inconsistent. The occurrence of multiple outliers suggests that DBSCAN variations' performance might vary greatly between runs, demonstrating their sensitivity to beginning circumstances and parameter adjustments. The increased variability reflects the algorithm's capacity to explore varied portions of the solution space, but it comes at the expense of decreased stability.

  The line graphs (figure 6) for the DBSCAN variants reveal a slower and more erratic convergence trend than GA. The cost values fluctuate more during iterations, suggesting that the algorithm regularly explores new parts of the solution space, which can result in major gains or degradations in solution quality. While extended exploration might aid in avoiding local optima, it also implies that the algorithm takes longer to stabilize and get near-optimal results. The changing pattern of the line plots indicates that DBSCAN is more exploratory, which may lead to higher quality solutions but needs more rounds to converge.

  DBSCAN variants are more computationally costly than GA and less stable. Their increased variability and slower convergence make them unsuitable for situations where computer resources are limited or consistent performance is essential. However, DBSCAN's ability to explore multiple solutions might be useful in very complicated or irregular solution spaces where other algorithms may fail to locate suitable answers. As a result, DBSCAN may be better suited to issues that require more exploration than rapid convergence.

- **K-Means Variants: (KMEANS1, KMEANS2)**

  The box plots (Figure 5) for KMEANS1 and KMEANS2 reveal that these algorithms are slightly more variable than GA but more consistent than the DBSCAN variations. The interquartile ranges are quite large, reflecting a balanced investigation of the solution space. The existence of fewer outliers than DBSCAN indicates that K-Means versions provide more consistent performance. KMEANS2 has a somewhat wider IQR than KMEANS1, indicating more comprehensive exploration capabilities, which can lead to better solutions but with higher unpredictability.

The line plots (Figure 6) for KMEANS1 and KMEANS2 indicate their convergence tendencies, with KMEANS1 converging similarly to GA, resulting in a smooth and consistent improvement in cost values. KMEANS2, despite having higher variability, tends to discover better solutions with more iterations, as seen by the slow but constant drop in cost values over time. The line plots show that, while KMEANS2 takes longer to converge, it investigates a broader range of solutions, which might be useful in identifying higher-quality solutions in difficult optimization problems.

When compared to GA, K-Means versions strike an appropriate balance between exploration and exploitation. KMEANS1 performs similarly to GA, giving it a feasible option for balancing computational economy and solution quality. KMEANS2, with its increased variability and processing needs, is better suited to situations where the end solution quality is more essential than convergence time. K-Means versions, notably KMEANS2, might be useful in cases that need a comprehensive investigation of the solution space to find optimum solutions.

- **Affinity Propagation (AP1, AP2)**

  The box plots (Figure 5) for AP1 and AP2 show considerable variability, with AP1 closest to GA in terms of consistency. The interquartile ranges for AP1 are quite small, showing consistent performance throughout several runs. AP2 has a larger IQR and more outliers, indicating that it is less consistent than AP1. The differences in variability between AP1 and AP2 represent the effect of parameter settings and clustering dynamics on performance.

  The line graphs (Figure 6) for AP1 and AP2 demonstrate separate convergence tendencies. AP1 converges fast, with a smooth and consistent reduction in cost values, comparable to GA. This demonstrates that AP1 efficiently balances exploration and exploitation, resulting in rapid convergence to near-optimal solutions. In comparison, AP2 has a more erratic convergence pattern, taking longer to settle and exhibiting more large changes in cost values. This shows that AP2 is more experimental, but it takes more iterations to get stable results.

When compared to GA, AP1 stands out as a viable option because to its rapid convergence and consistent performance, making it appropriate for medium and large datasets where efficiency is critical. AP2, with its increased variability and slower convergence, may require additional tweaking to improve efficiency. Affinity Propagation, particularly AP1, strikes a compromise between exploration and efficiency, making it a feasible solution for applications requiring speedy and consistent convergence. The performance of AP2 suggests that it may be improved by tweaking parameters and refining clustering algorithms.

- **Formal Concept Analysis (FCA1, FCA2)**

  The box plots (Figure 5) for FCA1 and FCA2 reveal more variability than GA, with FCA1 exhibiting moderate variability and FCA2 demonstrating the greatest variability of all algorithms. The larger interquartile ranges for FCA variations indicate that these algorithms investigate a greater range of answers, but with less consistency. The existence of several outliers in FCA2's box plot demonstrates that its performance can vary greatly across runs, indicating its exploratory character.

  The line graphs (Figure 6) for FCA1 and FCA2 demonstrate their convergence patterns, with FCA1 showing a consistent but variable improvement in cost values. While FCA1 converges quite well, it has more variance than GA, showing that it investigates several possibilities but takes longer to stable. FCA2, on the other hand, has a highly variable convergence tendency with large ups and downs, indicating that it is more exploratory but less stable in finding consistent answers.

  Compared to GA, FCA variations provide a more exploratory approach, which may lead to higher-quality solutions but with larger unpredictability and computing demands. FCA1, with its competitive CPU speeds and mild variability, is a suitable option where some variation in answers is acceptable. However, FCA2 requires more refining to increase its stability and efficiency. The use of Formal Concept Analysis in these algorithms demonstrates their potential to explore complicated solution spaces, although optimization and parameter adjustment are required to make them more practical for bigger datasets.

- **Mean Shift (MS1, MS2)**
  The box plots (Figure 5) of MS1 and MS2 reveal that MS1 has low variability and excellent consistency, making it equivalent to GA. MS1's small interquartile range and negligible outliers imply consistent performance throughout several runs. MS2 has more variability, a broader IQR, and more outliers, indicating less consistent performance. The variations in variability between MS1 and MS2 demonstrate how parameter selections affect their capacity to explore and converge.
  The line plots (Figure 6) for MS1 and MS2 corroborate their convergence patterns, with MS1 exhibiting a smooth and efficient decrease in cost values across iterations. This shows that MS1 efficiently balances exploration and exploitation, resulting in a rapid and consistent convergence to optimal solutions. MS2 shows a more variable convergence trend with considerable variability in cost values, implying that it is more exploratory yet takes longer to settle and obtain consistent solutions.
  MS1 outperforms GA in terms of low variability and efficiency, making it a good competitor for cost maximization tasks. MS2's increased unpredictability and longer CPU durations suggest possible inefficiencies, making it unsuitable for applications demanding consistent and rapid convergence. The Mean Shift method, notably MS1, provides a strong alternative to GA by combining stability and efficiency while exploring and improving the solution space. MS2, while more exploratory, requires more tweaking to improve its usefulness and speed.

DBSCAN clustering versions, notably DBSCAN2, perform poorly when compared to the Genetic Algorithm (GA). The boxplots (figure 5) for DBSCAN2 show substantially more variability and a larger number of outliers, indicating uneven performance across runs. The greater interquartile ranges represent the algorithm's extensive exploration of the solution space, but at the expense of stability and predictability. The line graphs highlight DBSCAN2's inefficiencies, with slower convergence rates and more large swings in cost values across iterations. This suggests that, while DBSCAN2 can explore a variety of solutions, it takes longer to stabilize and frequently requires more processing resources. DBSCAN2's high CPU times make it infeasible for cases requiring computational efficiency and constant performance. In comparison to GA, DBSCAN2's slow convergence to stable solutions and significant computing needs emphasize its shortcomings. The algorithm's sensitivity to beginning circumstances and parameter adjustments adds to its unpredictability, making it less trustworthy for cost maximization applications. Overall, DBSCAN2's vast exploration capabilities may find a variety of solutions, but its slowness and lack of consistency make it the least successful approach in our comparative comparison.

Among all of the methods tested, the Mean Shift clustering version, specifically MS1, outperforms the Genetic Algorithm (GA). MS1 consistently shows low variability and excellent consistency across all sample sizes, as shown by narrow interquartile ranges and few outliers in its box plots. The line plots emphasize MS1's efficiency by displaying a smooth and constant convergence trend comparable to GA. MS1's ability to quickly stabilize and achieve near-optimal solutions, along with its reasonable CPU times, makes it a strong competitor. This stability and efficiency are critical for cost maximization jobs that need solution consistency and computational efficiency. The combination of Mean Shift clustering and GA allows MS1 to successfully search the solution space while striking a balance between exploration and exploitation. This balance means that MS1 may avoid local optima while producing high-quality solutions in a manner similar to GA. MS1's resilience and scalability across diverse dataset sizes demonstrate its adaptability and efficacy, establishing it as the superior option to GA for combinatorial optimization issues.

6. Conclusion and future works

This work carefully investigated the performance of several hybrid genetic algorithms (GAs) combined with different clustering strategies for solving combinatorial optimization problems, with a special emphasis on the Knapsack problem. We conducted thorough experimental study to evaluate the performance of the basic Genetic Algorithm to its variations when paired with DBSCAN, K-Means, Affinity Propagation, Mean Shift, and Formal Concept study (FCA)-based clustering approaches.

The results show that the Genetic Algorithm with Mean Shift Clustering (MS1) consistently outperforms other versions including the conventional GA across a range of sample sizes. MS1 displayed minimal variability and great stability, making it ideal for cost maximization jobs requiring constant and efficient convergence. In contrast, DBSCAN variations, particularly DBSCAN2, demonstrated high unpredictability and computational inefficiency, rendering them unsuitable for scenarios needing speedy and reliable performance.

K-Means variations (KMEANS1 and KMEANS2) provided a mix of exploration and computing efficiency, with KMEANS2 exhibiting greater unpredictability but the potential to find high-quality solutions. Affinity Propagation (AP1) and FCA-based clustering (FCA1) demonstrated modest variability and competitive performance, indicating possibility for additional parameter adjustment.

Overall, combining clustering techniques with GAs improves solution space search, but efficiency and stability varies greatly amongst clustering approaches. MS1 emerged as the most reliable and economical technique, demonstrating Mean Shift clustering's promise in improving evolutionary algorithms for combinatorial optimization issues.

To address the drawbacks discovered in particular algorithms and further improve the performance of hybrid genetic algorithms, various detailed enhancements might be recommended for further research:

- **Adaptive Parameter Tuning:** One significant area for improvement is the dynamic adjustment of parameters such as crossover rate, mutation rate, and clustering criteria. Using adaptive parameter tuning approaches, such as self-adaptive genetic algorithms or reinforcement learning-based methods, the algorithm may fine-tune its parameters in real time based on the current state of the optimization process. This can assist algorithms like DBSCAN and FCA improve their stability and performance across multiple datasets by adapting to changing issue environments.
- **Hybrid Clustering Approaches:** Creating hybrid clustering methods that incorporate the best features of several clustering algorithms can result in more effective optimization. For example, combining Mean Shift clustering with K-Means or Affinity Propagation can leverage Mean Shift's capacity to handle complicated data distributions and K-Means' computational efficiency. Future study might look into algorithms that dynamically switch between clustering methods or utilize ensemble techniques to combine the capabilities of many clustering approaches.
- **Parallel and Distributed Computing:** To address the high computational needs seen in some variations, notably DBSCAN and MS2, parallel and distributed computing frameworks can be used. Implementing these methods on parallel architectures, such as GPUs or distributed systems, can greatly reduce computing time. Future research might concentrate on improving these algorithms for parallel execution, assuring effective load balancing, and reducing inter-process communication cost.
- **Machine Learning-Based Initialization:** Using machine learning models to initialize the population in genetic algorithms can give a more educated and strategic starting point, hence increasing convergence speed and solution quality. Techniques such as supervised learning, which predicts excellent beginning solutions based on previous data, and unsupervised learning approaches like clustering, which find promising portions of the solution space, might be investigated. Reinforcement learning agents might potentially be employed to direct the initialization process by learning the best tactics via trial and error.

- **Improved Exploration Mechanisms:** Using advanced exploration mechanisms can minimize premature convergence and increase the diversity of solutions. Multi-objective optimization, which involves optimizing many competing objectives at the same time, can be especially successful. This strategy can assist to preserve genetic variety and explore a broader range of options. Techniques such as Pareto optimization and evolutionary techniques that aim to preserve a varied collection of solutions in the population should be explored.
- **Real-World Application Testing:** Applying the improved algorithms to a number of real-world situations in various disciplines can offer a better understanding of their practical value and opportunities for improvement. Real-world applications in logistics, economics, and bioinformatics, such as gene selection and protein structure prediction, may be used to evaluate these methods. Such applications can identify unique obstacles and possibilities, directing future research toward more effective and domain-specific solutions.
- **Algorithm Robustness and Scalability:** It is critical to ensure that algorithms are both resilient and scalable across a wide range of issue sizes and kinds. Future research should focus on establishing techniques to improve algorithm resilience, such as noise and outlier tolerance, as well as ensuring scalability to large-scale challenges. Hierarchical clustering and divide-and-conquer algorithms can be used to efficiently manage large datasets.
- **Algorithm Benchmarking and Comparative Analysis:** Comparing the upgraded algorithms to a wide range of common optimization problems and datasets might reveal useful information about their strengths and flaws. Comparative study using cutting-edge optimization approaches will assist in determining the relative performance of the proposed additions and directing future developments.

By focusing on these specific areas of future study, researchers might improve the creation of more efficient, resilient, and scalable hybrid genetic algorithms capable of handling complicated combinatorial optimization problems in a variety of real-world contexts.

7. References

1. Carreira-Perpinán, M. A. (2015). A review of mean-shift algorithms for clustering. arXiv preprint arXiv:1503.00687.
2. Wang, K., Zhang, J., Li, D., Zhang, X., & Guo, T. (2008). Adaptive affinity propagation clustering. arXiv preprint arXiv:0805.1096.
3. Creţulescu, R. G., Morariu, D. I., Breazu, M., & Volovici, D. (2019). DBSCAN algorithm for document clustering. International Journal of Advanced Statistics and IT&C for Economics and Life Sciences, 9(1), 58-66.
4. Khan, K., Rehman, S. U., Aziz, K., Fong, S., & Sarasvady, S. (2014, February). DBSCAN: Past, present and future. In The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014) (pp. 232-238). IEEE.
5. Likas, A., Vlassis, N., & Verbeek, J. J. (2003). The global k-means clustering algorithm. Pattern recognition, 36(2), 451-461.
6. Khuri, S., Bäck, T., & Heitkötter, J. (1994, March). An Evolutionary Approach to Combinatorial Optimization Problems. In ACM Conference on Computer Science (pp. 66-73).
7. Misevičius, A., Blažauskas, T., Blonskis, J., & Smolinskas, J. (2004). An Overview of Some Heuristic Algorithms for Combinatorial Optimization Problems. Information Technology and Control, 30(1).
8. Hristoskova, A., Boeva, V., & Tsiporkova, E. (2014). A formal concept analysis approach to consensus clustering of multi-experiment expression data. BMC bioinformatics, 15, 1-16.
9. Sumangali, K., & Ch, A. K. (2019). Concept lattice simplification in formal concept analysis using attribute clustering. Journal of ambient intelligence and humanized computing, 10(6), 2327-2343.

10. Poelmans, J., Kuznetsov, S. O., Ignatov, D. I., & Dedene, G. (2013). Formal concept analysis in knowledge processing: A survey on models and techniques. Expert systems with applications, 40(16), 6601-6623.

11. Caro-Contreras, D. E., & Mendez-Vazquez, A. (2013, October). Computing the Concept Lattice using Dendritical Neural Networks. In CLA (pp. 141-152).

12. Wollbold, J., Guthke, R., & Ganter, B. (2008). Constructing a knowledge base for gene regulatory dynamics by formal concept analysis methods. In Algebraic Biology: Third International Conference, AB 2008, Castle of Hagenberg, Austria, July 31-August 2, 2008 Proceedings 3 (pp. 230-244). Springer Berlin Heidelberg.

13. Sohrabi, M., Fathollahi-Fard, A. M., & Gromov, V. A. (2023). Genetic Engineering Algorithm (GEA): An Efficient Metaheuristic Algorithm for Solving Combinatorial Optimization Problems.

14. Harik, G. R., & Lobo, F. G. (1999, July). A parameter-less genetic algorithm. In GECCO (Vol. 99, pp. 258-267).

15. Haldurai, L., Madhubala, T., & Rajalakshmi, R. (2016). A study on genetic algorithm and its applications. International Journal of computer sciences and Engineering, 4(10), 139.

16. Mathew, T. V. (2012). Genetic algorithm. Report submitted at IIT Bombay, 53.

17. Schmitt, L. M. (2001). Theory of genetic algorithms. Theoretical Computer Science, 259(1-2), 1-61.

18. Chehouri, A., Younes, R., Khoder, J., Perron, J., & Ilinca, A. (2017). A selection process for genetic algorithm using clustering analysis. Algorithms, 10(4), 123.

19. Zeebaree, D. Q., Haron, H., Abdulazeez, A. M., & Zeebaree, S. R. (2017). Combination of K-means clustering with Genetic Algorithm: A review. International Journal of Applied Engineering Research, 12(24), 14238-14245.

20. Feng, Z. (2012). Data clustering using genetic algorithms. Evolutionary computation: project report, CSE484.

21. Bezdek, J. C., Boggavarapu, S., Hall, L. O., & Bensaid, A. (1994, June). Genetic algorithm guided clustering. In Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence (pp. 34-39). IEEE.

22. Chakraverty, S., Sachdeva, A., & Singh, A. (2014, May). A genetic algorithm for task allocation in collaborative software development using formal concept analysis. In International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014) (pp. 1-6). IEEE.

23. Aziz, A. S. A., Salama, M. A., ella Hassanien, A., & Hanafi, S. E. O. (2012). Artificial immune system inspired intrusion detection system using genetic algorithm. Informatica, 36(4).

24. Cole, R. (2000, January). Automated layout of concept lattices using force directed placement and genetic algorithms. In Proceedings 23rd Australasian Computer Science Conference. ACSC 2000 (Cat. No. PR00518) (pp. 31-42). IEEE.

25. Aziz, A. S. A., Salama, M., ella Hassanien, A., & Sanaa, E. L. (2012, September). Detectors generation using genetic algorithm for a negative selection inspired anomaly network intrusion detection system. In 2012 Federated Conference on Computer Science and Information Systems (FedCSIS) (pp. 597-602). IEEE.

26. Cintra, M. E., Monard, M. C., & Camargo, H. A. (2015, August). FCA-based rule generator, a framework for the genetic generation of fuzzy classification systems using formal concept analysis. In 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (pp. 1-8). IEEE.

27. Cacchiani, V., Iori, M., Locatelli, A., & Martello, S. (2022). Knapsack problems—An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems. Computers & Operations Research, 143, 105693.

28. Pisinger, D. (1999). Core problems in knapsack algorithms. Operations Research, 47(4), 570-575.

29. Caprara, A., Pisinger, D., & Toth, P. (1999). Exact solution of the quadratic knapsack problem. INFORMS Journal on Computing, 11(2), 125-137.

30. Pisinger, D. (2005). Where are the hard knapsack problems?. Computers & Operations Research, 32(9), 2271-2284.

31. Kostenetskiy, P. S., Chulkevich, R. A., & Kozyrev, V. I. (2021). HPC resources of the higher school of economics. In Journal of Physics: Conference Series (Vol. 1740, No. 1, p. 012050). IOP Publishing.