Wiley
International Journal of Intelligent Systems
Volume 2024, Article ID 5571499, 33 pages
https://doi.org/10.1155/2024/5571499

Research Article

An Innovative Application of Swarm-Based Algorithms for Peer Clustering

Vesna Šešum-Čavić,[1] **Eva Kühn,**[2] **and Laura Toifl**[2]

[1]*University of Belgrade, Faculty of Civil Engineering, Bulevar kralja Aleksandra 73, Beograd 11000, Serbia*
[2]*TU Wien, Faculty of Informatics, Argentinierstr. 8, Vienna 1040, Austria*

Correspondence should be addressed to Vesna Šešum-Čavić; vsesumcavic@grf.bg.ac.rs

Received 29 March 2024; Revised 16 August 2024; Accepted 28 August 2024

Academic Editor: Said El Kafhali

In most peer-to-peer (P2P) networks, peers are placed randomly or based on their geographical position, which can lead to a performance bottleneck. This problem can be solved by using peer clustering algorithms. In this paper, the significant results of the paper can be described in the following sentences. We propose two innovative swarm-based metaheuristics for peer clustering, slime mold and slime mold K-means. They are competitively benchmarked, evaluated, and compared to nine well-known conventional and swarm-based algorithms: artificial bee colony (ABC), ABC combined with K-means, ant-based clustering, ant K-means, fuzzy C-means, genetic K-means, hierarchical clustering, K-means, and particle swarm optimization (PSO). The benchmarks cover parameter sensitivity analysis and comparative analysis made by using 5 different metrics: execution time, Davies–Bouldin index (DBI), Dunn index (DI), silhouette coefficient (SC), and averaged dissimilarity coefficient (ADC). Furthermore, a statistical analysis is performed in order to validate the obtained results. Slime mold and slime mold K-means outperform all other swarm-inspired algorithms in terms of execution time and quality of the clustering solution.

Keywords: peer clustering; slime mold; swarm intelligence; unstructured P2P

1. Introduction

In the Internet, there are many popular file-sharing applications like Gnutella [1] or Napster [1], which were designed as P2P networks [2]. Established patterns like load balancing [3] or load clustering [4] can be used in such a system, when suffering from high loads. Unfortunately, these patterns are not ideal, if the positioning of the peers is the performance bottleneck. This is not unusual and it results in facing significant problems regarding the overall performance of the system because in most P2P networks, peers are placed randomly or based on their geographical position [5]. So, the performance in such networks can be extremely poor. The problem can be solved by peer clustering, i.e., grouping peers, which has certain characteristics in common, together as neighbors [4, 6].

However, some characteristics of P2P systems [2] make clustering a challenging task: (1) although peers in a P2P system are autonomous, autonomy is violated by data clustering since peers are enforced to store some specific data; (2) the very dynamic nature of P2P environments implies that clusters formed need to dynamically adapt to the frequent changes. The lack of global knowledge of data and peer interests also causes a serious difficulty in forming clusters in P2P systems. Due to the fact that peers are leaving and entering the network dynamically, also peer clustering has to be a dynamic procedure. With peer clustering, query performance can be significantly improved compared to a random network topology. That means requests are routed more efficiently and only to nodes which are likely to fit the request. Besides, if a cluster containing a node, which is likely to fit the request, can be found, query flooding through the whole network is not necessary. Consequently, this means that workload on nodes, which are probably not fitting the request, can be reduced [6].

High dynamics in the P2P environments emphasize the necessity of finding an intelligent, adaptable, and robust solution. Therefore, the problem of *peer clustering* is very specific as it should encompass all issues numbered above that characterize the environment. So, it requires advanced, intelligent, and self-organized approaches that are capable to cope with dynamics in the system.

In this paper, we propose an *intelligent peer clustering* in a *fully unstructured* P2P overlay network. In such environment, dynamic processes are extremely emphasized (e.g., nodes can spontaneously join or leave).

For the benchmarking process, a set of clustering algorithms is implemented: artificial bee colony (ABC), ABC combined with K-means, ant-based clustering, ant K-means, fuzzy C-means, genetic K-means, hierarchical clustering, K-means, and particle swarm optimization (PSO) together with the proposed slime mold and slime mold K-means are plugged in and compared. These algorithms are chosen because of the following reasons: (1) Dd-slime mold algorithm is a part of our previous work with promising results [8, 9]. (2) Ant-based and bee-based algorithms as well as PSO are popular swarm-intelligent algorithms [10]. (3) Genetic K-means [11] and fuzzy C-means [12, 13], hierarchical clustering [14, 15], K-means [16, 17] are typical conventional approaches. Additionally, all selected algorithms have a satisfactory part of the theoretical basement (e.g., clear mathematical modeling) [9]. The evaluation part at the end of this work provides a comparison of these clustering algorithms and is expected to show which algorithms perform particularly well in certain scenarios.

Slime mold intelligence is a novel approach for peer clustering. A motivation of choice of this intelligent approach is based on following reasons: (1) in our previous work, we obtained promising and satisfactory results employing this type of intelligent approach [7, 8]; (2) according to our experiences working with this type of intelligence, it addresses successfully also the "corner" cases, which are sometimes "neglected" applying other types of intelligence; and (3) the slime molds compute the optimal solution for the amount of resources involved [17].

Therefore, the research questions are as follows.

Can swarm intelligence effectively cope with peer clustering in fully unstructured P2P networks? How efficiently slime mold intelligence solves the problem of peer clustering in fully unstructured P2P networks?

1.1. Contribution of the Paper. Before presenting the contribution of the paper, we provide a kind of support[2] to help readers progress through the paper.

1.1.1. Regarding Methodology. Our methodology used can be explained schematically as shown in Figure 1. That is, we treat a problem by using a multidisciplinary approach that includes mathematical methods, interdisciplinary methods, and software engineering methods.

1.1.1.1. M1: Mathematical Models. After the analysis of swarm mechanisms in nature, the behaviors that exist in a respective swarm mechanism and govern natural phenomena must be described using mathematical representations as the abstraction of the reality, i.e., by using stochastic modeling and discrete mathematics tools.

1.1.1.2. M2: Mathematical Proofs. The algorithms themselves must be **analyzed** and their behavior explained and theoretically proven (this part is out of scope of this paper).

1.1.1.3. I1: Swarm-Inspired Algorithms (SW-Inspired). Swarm-inspired **algorithms** appear as a consequence of used mathematical models. For each investigated swarm mechanism in nature, one must analyze how it can be mapped to IT terminology. In mapping, software agents play the role of a particular swarm and perform self-organized actions characteristic for the respective swarm colony.

1.1.1.4. I2: Parameters and Metrics. The so-called best parameter settings must be derived for each algorithm for each considered problem scenario, as all swarm inspired algorithms are characterized by a huge number of different environmental parameters influencing the behavior of artificial swarms. Evaluation criteria and specification of suitable metrics for scalability, performance, quality of solution, etc. must be identified.

1.1.1.5. I3: Impact Analysis. A feedback loop to improve the mathematical models based on the benchmark results is suggested.

1.1.1.6. SE1: Algorithms. The newly invented algorithms must be implemented based on the mathematical and algorithmic description (see I1).

1.1.1.7. SE2: Coordination Generics. Reusable generic coordination mechanisms must be extracted for each use case as "patterns" which can be **understood** as universal blue prints. **They** must be implemented as reusable components. Their designs shall be inspired by multiagent technologies.

1.1.1.8. SE3: Frameworks for Self-Organization. A generic framework with self-organizing properties as a **composition** of the components **described** above must be designed and developed. The framework allows the exchange of different, swarm-inspired as well as other algorithms simply through "plugging" and must support many different network topology settings through configuration. The plugging approach is achieved by a component-based design, where each agent represents a certain exchangeable policy or behavior. This allows fair comparison of different algorithms in a neutral framework.

1.1.1.9. SE4: Test Bench. A test environment for automatically running and interpreting benchmarks in all **desired** framework configurations and algorithm combinations must be developed.

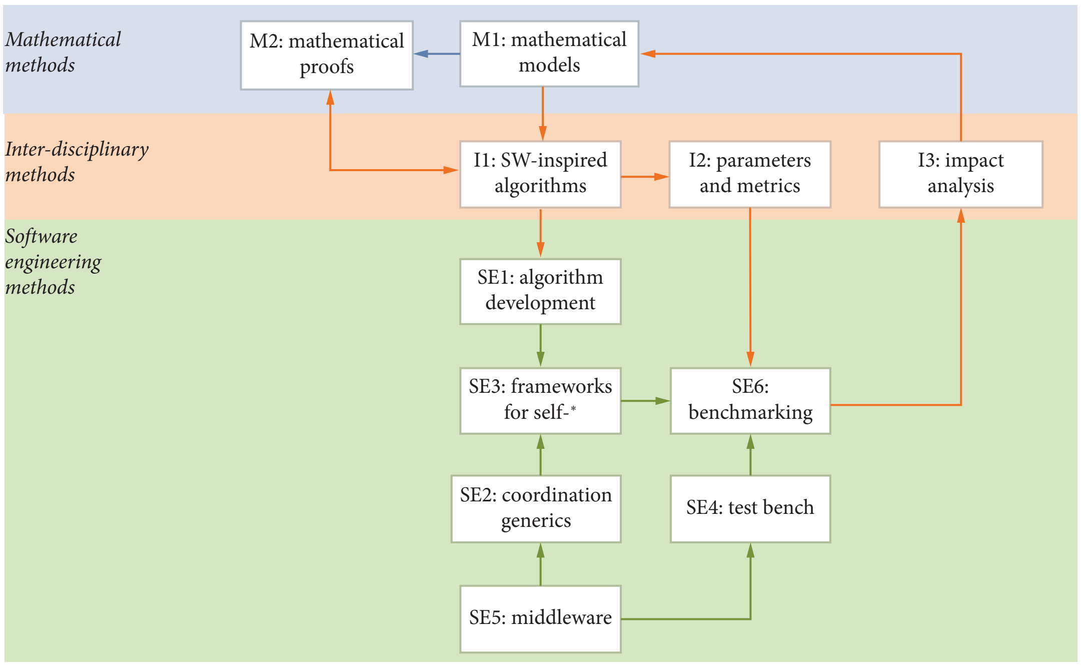
Mathematical methods
M2: mathematical proofs
M1: mathematical models
Inter-disciplinary methods
I1: SW-inspired algorithms
I2: parameters and metrics
I3: impact analysis
Software engineering methods
SE1: algorithm development
SE3: frameworks for self-*
SE6: benchmarking
SE2: coordination generics
SE4: test bench
SE5: middleware

Figure 1: Interplay of mathematical (M), software engineering (SE), and interdisciplinary (I) methods. *Note:* "Self- * " means different aspects of self-organizing properties such as self-healing, self-repairing, and self-configuring [18].

1.1.1.10. SE5: Middleware. In order to achieve SE2 and SE4, suitable middleware architecture must be selected.

1.1.1.11. SE6: Benchmarking. Finally, the benchmarks need to be performed and the results evaluated using the parameters and metrics of I2.

Note: Recall that out of scope of this paper are the following from Figure 1: mathematical proofs (M2) and explanation of the framework used and the middleware (SE2, SE3, and SE5). Recognize that the main focus is on the swarm-inspired algorithms (I1 as well as I2 and I3), accompanied mathematical models (M1) and the algorithm development (SE1), and consequently on the benchmarking (SE4 and SE6).

1.1.2. Regarding the Problem at Hand. Recall that in the scope of this paper, we consider an intelligent peer clustering in a fully unstructured P2P overlay network.

1.1.3. Regarding the Swarm-Based Algorithms (I1) Applied. We applied two innovative intelligent algorithms for peer clustering based on the slime mold life cycle and also we further adapted 5 swarm-based algorithms.

For better understanding the slime mold life cycle, we provide a short biological background of slime mold in nature.

The slime mold *Dictyostelium discoideum* (Dd) is a social collective of self-organizing amoebas that goes through a life cycle. The goal of Dd amoebas is their feeding and therefore a constant supply of food that consists of bacteria and decaying material in the soil. The Dd life cycle has several stages (Figure 2): vegetative movement, aggregation, mound, slug movement, fruiting body formation, and spore dispersal [19].

In the vegetative movement, amoebas navigate through the soil on their own, using tentacle-like pseudopods, and search for food. Furthermore, in this state of adequate nourishment, amoebas may procreate by fission. An amoeba stays in this state until the food supply shrinks. If not enough food supply is available, an amoeba begins to starve [19]. At this point, the collaboration of amoebas begins and the aggregation stage of the life cycle starts. Amoebas in this stage communicate indirectly by emitting a pheromone called cyclic adenosine monophosphate (cAMP). More specifically, one of the amoebas takes the role of the pacemaker, which releases cAMP and therefore causes all others to be drawn to it until a mound is formed [19]. When aggregating to a mound, the amoebas emit a slimy substance. In the mound, amoebas start to organize themselves into two categories, prespore and prestalk, based on their fitness level. The fitness level is determined on the basis of an amoeba's level of nourishment, i.e., how efficient collecting food in the vegetative stage was. Prespore amoebas move to one end of the mound and form the head, while the prestalk amoebas form the slime mold tail [19]. At the end of this aggregation process, the mound of amoebas has formed a slug and starts moving to a source of light. The amoebas try to reach the surface of the soil until they die [19]. If they succeed to reach the surface, amoebas start organizing themselves into a fruiting body. The least fit amoebas, which formed the tail of the mound, sacrifice themselves and die after forming the stalk of the fruiting body. After that, the head amoebas climb the stalk and transform into spores. Spores are dispersed by

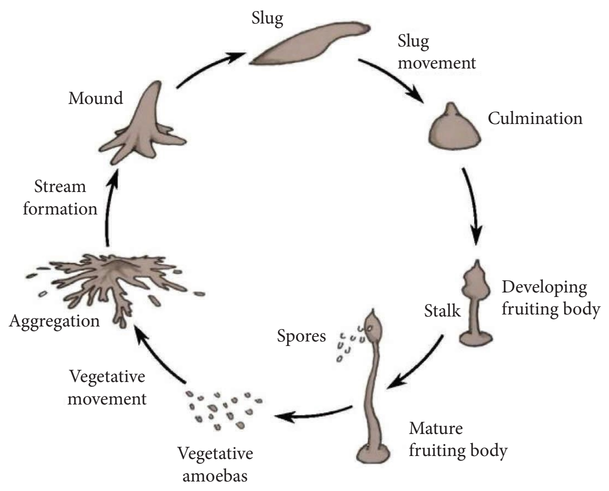

Figure 2: Life cycle of *Dictyostelium discoideum* [19].

environmental factors such as animals or the wind. After the dispersal and the process of germination, spores become active amoebas and the life cycle begins again at the vegetative stage [19].

Recall that the objective of the paper is to answer the research questions presented in the first part of Section 1 (before Subsection 1.1).

So, to revise, the *novelty* and *contribution* of this paper include the following:

- Two cutting-edge intelligent algorithms for peer clustering, inspired by the life cycle of the Dd slime mold, are presented: slime mold and slime mold K-means.
- Adaptation of 5 swarm-based algorithms for the problem of peer clustering.
- A “fair” comparison (based on the peer model framework) between selected algorithms: ABC, ABC combined with K-means, ant-based clustering, ant K-means, fuzzy C-means, genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means.

This paper is structured as follows. In Section 2, the related work is discussed according to the proposed methodology used in the scope of this research. Section 3 covers the implemented swarm-inspired peer clustering algorithms. In Section 4, the benchmark methodology is defined and the benchmark results are evaluated. Finally, Section 5 discusses possible future improvements and contains the conclusion. The Appendix contains the raw data results.

2. Related Work

The target area of the state of the art covers approaches and algorithms exclusively for *peer clustering* in P2P networks. In the following section, different approaches for clustering peers are discussed.

In [20], a semantic overlay network (SON) is created. For this purpose, peers with similar contents, like music genre, are connected to each other, consequently building a semantic cluster. Therefore, the peers, the queries, and the documents themselves, respectively, have to be classified in order to determine to which cluster(s) they belong. Thus, queries can directly be routed to the respective SON, actually improving the search performance.

In [21], the authors proposed a model for forming groups of peers implicitly, called communities, based on common interests. Those communities are formed as peers claim their interests analogous to social networks and are possibly overlapping. A peer’s interests can be provided explicitly by the peer or implicitly identified from past queries. Furthermore, they present a search technique, which is based on the ability of the peers to form communities.

In [5], a similar model like in [21] is introduced, but here for the interests, a predefined ontology is used instead of letting the peers claim their interests freely. This makes it easier for a peer to find other peers with similar interests. The authors proposed a model that uses social network concepts as the main physical structure for clustering peers. Nodes in the model are grouped into several communities and subcommunities with similar interests. The order and relation among communities are defined and controlled by a shared ontology.

In [22], a peer’s interests are found out by extracting keywords from text documents in their storage. The keyword extraction affects the global keyword vocabulary and is treated as a decision problem. Although this is an interesting approach, it is only applicable for text documents.

In [6], two algorithms are proposed. They aim at creating an overlay network by clustering similar peers and are based on Schelling’s model. One unique property is used to cluster peers having this property in common, and thus they do not overlap. Schelling’s model explains the existence of segregated neighborhoods in America. It consists of a 2-dimensional grid where two thirds of the cells are randomly populated with blue and red turtles. The turtles wish to have at least a certain percentage of neighbors having the same color as them. If this is not the case, the turtle moves to an adjacent cell. This continues until all turtles are satisfied with their neighbors.

In [23], the authors proposed a cluster-based P2P system, called PeerCluster, for sharing data over the Internet. In PeerCluster, all participant computers are grouped into various interest clusters, each of which contains computers that have the same interests.

Further, papers [24, 25] investigate forming clusters by using P2P networks, but they are focused specifically on electricity and energy markets, i.e., very problem-specific oriented.

However, all existing approaches are very problem specifically oriented or take in consideration only some properties of the system, and it would be difficult to apply them on the general problem of peer clustering. In order to analyze abovementioned related work approaches, Table 1 is created. A table encompasses necessary properties required in the evaluation methodology [18]:

- Provisioning of a general framework[3]: As there is no "one-fits-all" solution, in order to find the best algorithm for a use case, a generalized framework is needed that allows the testing and tuning of different algorithms for a specific use case and environment. We differentiate between the complete solution for a domain specific use case and the algorithm(s) integrated/used by the solution. A framework does not per se solve the entire use case but serves as a necessary base for the spectrum of algorithms used.
- Composability of the architecture: The architecture must be flexible, so that neither new requirements on specific algorithms nor other assumptions on the network infrastructure become "architecture breakers." The flexible exchange of components and algorithms as well as combinations of different components within the framework shall be possible.
- Autonomy and self-organizing properties: Intelligent algorithms require agents as they are advantageous in situations that are characterized by high dynamics, unforeseeable events, and heterogeneity. Use cases with huge complexity, diversity of requirements, and dynamically changing configurations force identifying new solutions based on self-organization, autonomic computing, and mobile agents.
- Support of arbitrary configurations: It targets problem and domain independency that allow arbitrary network topologies.
- Benchmarking in different environments: Benchmarking shall be possible in different environments providing a "fair" comparison of presented approaches/algorithms.
- Intelligent algorithm's support: As already written, high dynamics in the P2P environments emphasize the necessity of finding an intelligent, adaptable, and robust solution.

3. Swarm-Inspired Algorithms for Peer Clustering

Swarm-based algorithms are inspired by nature and aim at imitating the behavior of life forms organized in swarms and apply it to the solution of problems. The most widely used groups of swarm-inspired clustering algorithms are ant-based algorithms, closely followed by bee-inspired algorithms. Similar to P2P systems, swarms are also perceived to be decentralized and self-organized systems. Thus, swarm-inspired algorithms are likely to provide satisfying results.

At the beginning of this section, the problem formulation is given. That is, as the first step, a model for the P2P lookup requirements is described as it is necessary further to define the problem formulation. Also, the criteria for clustering are introduced.

Later, the swarm-inspired algorithms used for peer clustering are described. Thus, after a general, abstract explanation of an algorithm, its mapping and adaptation to the peer clustering problem are discussed. Two innovative algorithms for peer clustering based on the slime mold life cycle are presented in Subsections 3.9 and 3.10. At the end of these subsections, their main characteristics are listed.

3.1. P2P Resource.

3.1. P2P Resource. A resource in the P2P system can be defined as a combination of content and its metadata. For reasons of simplicity, only the file name is taken into account for the formal definition. Therefore, r stands for a *file* resource and q stands for a *query request*. In the following, we define the storage peer as a peer storing data and processing data queries and the worker peer as a peer accepting and performing job requests like computations.

Sample instances of a request and two file resources are shown below.

Query Request: ("filename.txt")

Matching File Resource: ("filename.txt")

Non-Matching File Resource: ("file.txt")

In contrast to a file resource, a job is no concrete resource. It is a metaphorical concept, defined as a combination of a job type, a descriptive categorization of the job, and a computational power. Nevertheless, it defined as a tuple $j = (j_1, j_2)$, where j_1 depicts the job categories and j_2 is the currently available computational power, i.e., the available resources. A *job request* is structured in the same manner. Therefore, a worker peer doing job j^r can only accept a job request j^q if and only if $j_1^q \in j_1^r \wedge j_2^r \geq j_2^q$.

Table 1: Necessary properties required in the evaluation methodology: "x," the feature is supported; "–," the feature is not supported.

	Provisioning of a general framework	Composability of the architecture	Autonomy and self-organizing properties	Support of arbitrary configurations	Benchmarking in different environments	Intelligent algorithms' support
[5]	—	—	x	—	—	—
[6]	—	—	x	—	—	—
[20]	—	—	x	—	—	—
[21]	—	—	x	—	x	—
[22]	—	—	—	—	x	—
[23]	—	—	x	—	x	—

In order to clarify the formalism above, an illustrative example is provided by presenting sample instances of a job resource and three job requests.

Job Resource: ("sorting, optimization", 3.2)
Acceptable Job Request: ("sorting", 2.3)
Non-Acceptable Job Request: ("hash value calculation", 2.3)
Non-Acceptable Job Request: ("optimization", 3.3)

3.2. Clustering Criteria. Different criteria are used for clustering storage and worker peers. For storage peers, the main criterion is the topic interest. For reasons of simplicity, it is used the file name of the respective file resource for that purpose. The more topic interests two peers share, the higher is the probability of sharing the same cluster. For reasons of clarification, an example is given below.

Storage Peer 1: ("sorting, optimization")
Storage Peer 2: ("sorting")
Storage Peer 3: ("hash value calculation")

In the example, storage peer 1 and storage peer 2 would be sharing a cluster, due to the fact that they have a topic interest in common, whereas storage peer 3 would be placed in another cluster.

On the contrary, for worker peers, there are two main criteria for clustering: the job types processed by a worker peer and its currently available computational power. Thus, both of these criteria are taken into account in equal proportions when examining whether two peers are similar to each other or not. The below given example will illustrate this.

Worker Peer 1: ("sorting, optimization", 4.4)
Worker Peer 2: ("sorting", 4.3)
Worker Peer 3: ("sorting", 1.2)

In the example given above, worker peer 1 and worker peer 2 are very likely to share a cluster. Although worker peer 3 also shares the same job type, its probability of being part of the same cluster as worker peer 1 and worker peer 2 is not that high as its computational power is considerably lower.

3.3. Distance Measurement. Distance measurement is important for peer clustering in order to learn which cluster is most suitable for a certain peer. Thus, the measurement of how well a peer fits into a cluster, in most algorithms called fitness value, is based on distance measurement. All measurements are based on the Euclidean distance[4] formula, which measures the distance between two points p and q in an n-dimensional space. However, as components might not be numbers, equation (1) presents an adapted measurement called "mean square distances," MSD, where each dimension has its own distance function:

$$\mathrm{MSD} = \sqrt{\sum_i d_i\left(p_i, q_i\right)^2}, \tag{1}$$

where d_i is the distance along dimension i. The smaller the distance is, the more similar the two measurement objects are. Vice versa, the larger the distance between two measurement objects is, the more dissimilar they are.

In this paper, the measurement type is chosen by the preference of the respective algorithm, if there is given any.

To quantify the dissimilarity of strings, a variation of the Levenshtein distance [26] is used. The Levenshtein distance of strings and the numerical distance are expressed as a percentage, where the latter is based on the following equation[5]:

$$pd(p,q) = \frac{|p-q|}{p+q/2}. \tag{2}$$

3.3.1. P2P Distance. **P2P distance** is used to calculate the distance between two peers. **Peer-to-cluster distance** is used to calculate the distance between a peer and a cluster mean. A cluster mean is embodied by its three most frequent keyword occurrences combined with their frequency of occurrence and the average available computational power. **Cluster-to-cluster distance** is used to calculate the distance between two clusters.

3.3.2. Jaccard Distance. In addition to the abovementioned distance measurement, all types of distance measures were also implemented using the Jaccard distance, based on the Jaccard index. Furthermore, as the distance measurements are, just as the peer clustering algorithms, implemented in a Plug and Play manner, they are easily interchangeable. Also, in this type of distance measurement, the numerical distance is expressed as a percentage (equation (3)) The formula for the Jaccard distance[6] is given as

$$d_J(p,q) = 1 - \frac{|p\cap q|}{|p\cup q|}. \tag{3}$$

An illustration of distance calculations is provided in the Appendix.

3.4. ABC. The ABC algorithm [27] is based on the foraging behavior of honey bees. A possible solution is represented by a food source and the food source's nectar amount complies with the fitness or quality of the associated solution. In an artificial bee hive, the foraging tasks are divided between employed bees and onlooker bees. Employed bees go to the food sources visited by them before, whereas an onlooker bee waits for the employed bee, giving them quality information about the visited food source. Then, the onlooker bee chooses a food source to exploit on the basis of the nectar quality. The more nectar a food source offers, the larger is the probability to be chosen by an onlooker bee. In the ABC algorithm, the number of employed bees and onlooker bees equals the number of food sources. Therefore, each food source is visited by only one employed bee.

In each iteration step, each employed bee produces a modification of the current solution v_i depending on the local information [27]:

$$v_i = z_i + \phi(z_i - \mathrm{z_k}), \tag{4}$$

where z_i is the current solution and z_k is a randomly selected food source differing from z_i. ϕ is a random number between -1 and 1. After producing the new source, its nectar amount is tested. Therefore, the fitness of the solution is calculated the following way [27]:

$$\mathrm{fit}_i = \frac{1}{1+f_i}, \tag{5}$$

where f_i is the sum of all instances of MSD between an instance and the associated cluster center, divided by the number of instances. A greedy selection is applied to the newly produced solution and the current one, where the better one is kept in memory. After all employed bees have completed the search, the nectar information of the food sources is shared with the onlooker bees. Therefore, the probability value p_i has to be calculated for each solution [27]:

$$p_i = \frac{\mathrm{fit}_i}{\sum_{n=1}^{\mathrm{SN}} \mathrm{fit}_n}, \tag{6}$$

where SN is the number of food sources equaling the number of employed bees and onlooker bees.

Based on this probability value, each onlooker bee chooses one food source. The higher the probability value of a certain solution is, the higher is the chance to be chosen by an onlooker bee. Thus, if a solution has a very high probability value, it may be chosen by multiple onlooker bees. The onlooker bee then produces a new solution based on the selected food source using equation (4). The fitness value of this solution is computed by equation (5) and, just as in the employed bees phase, a greedy selection process is applied between the newly produced solution and the probabilistically chosen one. The richest food source shall be memorized across all iterations (see Algorithm 1).

The algorithm is *mapped* to the peer clustering problem the following way.

At the initialization, the given number of food sources *noSources*, i.e., solutions to the clustering problem, is created. This happens by randomly assigning the peers to be clustered to a given number of clusters *k*. This is done in such a way that *n* peers are assigned to each cluster, where *n* is the number of peers to be clustered. Then, the rest of the peers are assigned to a randomly chosen cluster. Afterward, in each iteration, the following procedure is performed. Each employed bee creates a new solution based on its own food source and a randomly chosen one. More specifically, for each peer in both solutions, the associated cluster is expressed as a numeric value. This makes it possible to insert these numeric values into equation (4), resulting in a cluster expressed as a numeric value which shall be associated to the corresponding peer. Thus, the composition of the newly created solution is such that this is done for each peer. Next, the fitness value for the bee's solution and new solution is calculated as described above. This also applies to the choice of the better solution and the calculation of the probability

```
Input: number of clusters $k$, number of food sources noSources, maximum number of iterations maxIteration
1   initialization;
2   for $j := 1$ to maxIteration do
3       for each employed bee do
4           produce new solution $v_i$;
5           calculate fitness value;
6           apply greedy selection;
7       end
8       calculate the probability values $p_i$ for the solutions;
9       for each onlooker bee do
10          select a solution depending on $p_i$;
11          produce new solution $v_i$;
12          calculate fitness value;
13          apply greedy selection;
14      end
15      memorize the best solution so far;
16  end
```

Algorithm 1: Artificial bee colony algorithm.

value pi for each food source. Eventually, based on the probability value, each onlooker bee chooses a solution. This means the higher the fitness value of a solution is, the higher is p_i, and consequently the higher is the probability to be chosen by an onlooker bee. Then, the onlooker bee's procedure is basically the same as that of the employed bee.

3.5. Combination of ABC Algorithm and K-Means (ABCK). The ABCK algorithm [28] is a combination of the artificial bee algorithm and classical K-means. Thus, the procedure is basically the same as the procedure of ABC, only differing in the fact that each solution of ABCK is locally optimized by the execution of K-means. According to [28], the K-means algorithm is highly dependent on the centroid initialization while being computationally light, whereas the ABC algorithm is rather time-consuming. Therefore, a combination of those two algorithms shall complement each other. The setting of the ABCK algorithm is basically the same as of the ABC algorithm, regarding employed bees, onlooker bees, and the food sources. The procedure differs from the original one by adding an additional K-means step. Every time a bee produced a modification of a solution v_i, in the employed bee phase as well as in the onlooker bee phase, the K-means algorithm is applied on this solution. Then, the solution is evaluated by means of a criterion called distortion, introduced by Armano and Farmani [28] and defined as follows:

$$E = \sum_{k=1}^{K} \sum_{z_i \in C_k} \left\| z_i - C_k \right\|^2, \tag{7}$$

where z_i is the ith instance belonging into cluster C_k. The distortion sums up the squared distance between all peers and their associated cluster centers, using the peer-to-cluster distance mentioned in Section 3.3. If the distortion is low, the clustering is good, as it measures the inner distances within each cluster and thus the distortion shall be minimized. Therefore, the solution having the smaller distortion is kept in memory. Also in this procedure, the best solution shall be memorized across all iterations (see Algorithm 2).

Due to the fact that this algorithm is very similar to the pure ABC algorithm, the *mapping* to the peer clustering problem is exactly the same as described above, with the only exception that each newly produced solution is locally optimized by applying K-means on it.

3.6. Ant-Based Clustering. In ant-based clustering [29], the ants act on a two-dimensional grid which is populated randomly with items. An ant is empowered to execute one of two actions: an unloaded ant can pick up an element lying on a field currently visited by the ant. An ant carrying an element can drop it on a free cell. These actions are decisions influenced by the ants' perception of the environment. In this algorithm, an item represents the object which shall be clustered. Thus, an item equates to a peer. In each iteration, an ant is selected randomly. If there is an item at the ant's current position and the ant is not carrying one, the selected ant can pick the item. If the location is free, a carried item can be dropped by the ant. The probability of picking and dropping, respectively, is influenced by the similarity of the peers placed in the surrounding area, clarified in equation (10).

The picking and dropping probabilities for a given position i are calculated the following way [29]:

$$p_{\text{pick}}(i) = \left(\frac{k_{\text{pick}}}{k_{\text{pick}} + f(i)} \right)^2, \tag{8}$$

and

$$p_{\text{drop}}(i) = \begin{cases} 2f(i), & \text{if } f(i) < k_{\text{drop}}, \\ 1, & \text{otherwise}, \end{cases} \tag{9}$$

```
Input: number of clusters k, number of food sources noSources, maximum number of iterations maxIteration.
1  initialization;
2  for $j := 1$ to maxIteration do
3    for each employed bee do
4        produce new solution $v_i$;
5        apply k-means on $v_i$;
6        calculate distortion;
7        apply greedy selection;
8    end
9    calculate the probability values $p_i$ for the solutions;
10   for each onlooker bee do
11      select a solution depending on $p_i$;
12      produce new solution $v_i$;
13      apply k-means on $v_i$;
14      calculate distortion;
15      apply greedy selection;
16  end
17  memorize the best solution so far;
18 end
```

Algorithm 2: ABC algorithm combined with K-means.

where k_{pick} and k_{drop} are constants and $f(i)$ is the neighborhood function of the current location i:

$$f(i) = \begin{cases} \frac{1}{d^2}\sum_j \left(1 - \frac{d(i,j)}{\alpha}\right), & \text{if } f(i) > 0, \\ 0, & \text{otherwise,} \end{cases} \tag{10}$$

where the sum extends over all locations j in the surrounding area of location i, $d(i, j)$ is the measurement of dissimilarity between the items located at cells i and j and therefore computed by using the P2P distance, α is a scaling constant, and d^2 equals the size of the local neighborhood of i to normalize the result (see Algorithm 3).

This algorithm is *mapped* to the peer clustering problem as follows. Firstly, peers and ants are randomly distributed on the grid. Then, the ants move the peers as described above, by picking them up and dropping them at another location. At the end of the algorithm, a cluster is formed by all peers which are horizontally, vertically, or diagonally adjacent to each other.

3.7. Ant K-Means.

The ant K-means algorithm [30] is based on combining ant colony optimization (ACO) with conventional K-means clustering. It is inspired by the natural behavior of ant colonies, as they leave a trail of pheromones to communicate with each other. Pheromones are left by an ant following a certain path. Thus, the more ants take the same trail, the more pheromones lay on the trail. Consequently, this particular path becomes more attractive for other ants and the shortest route can be obtained. Thus, the conventional K-means is modified in such a way that it utilizes the described behavior.

In each iteration, the pheromones of each path have to be updated, using the following formula by Kuo et al. [30]:

$$\tau_{ij} = (1 - p)\tau_{ij} + \frac{Q}{\text{TWCV}}, \tag{11}$$

where p is the pheromone decay parameter, Q is a constant, and total within cluster variation (TWCV) is the total within cluster variance, the sum of the squared distance of each peer to its respective cluster mean.

Each ant m assigns each peer i to a cluster mean j with a probability P_{ij}^m which is calculated as follows [30]:

$$P_{ij}^m = \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_c^k \tau_{ic}^\alpha \eta_{ic}^\beta}, \tag{12}$$

where α is the relative importance of the trail and β is the relative importance of the visibility η, which is the inverse of the distance between peer i and cluster center j, i.e., $\eta_{ij} = 1/d_{ij}$.

Thereafter, the cluster means are updated and the new TWCV has to be calculated. This procedure is repeated until the TWCV does not change anymore. If the new TWCV is smaller than the global best TWCV, it is replaced.

Next, a step called perturbation is performed in order to not to get stuck with a local minimum. Basically, in this step, every ant is newly initialized, just as described in the initialization step below (see Algorithm 4).

The algorithm is *mapped* to the peer clustering problem the following way: At the initialization step, the given number of ants *noAnts* is created. As each ant holds its own solution to the peer clustering problem, this includes the initialization of k cluster means by randomly assigning peers to it. Afterward, lay equal pheromone on each path. Then, in each iteration step, the pheromones of each path are updated and each peer is assigned to the cluster with the highest probability P_{ij}^m, as described above.

```
Input: step size of an ant stepsize, maximum number of iterations maxIteration, picking constant $k_{pick}$, dropping constant $k_{drop}$, scaling parameter $\alpha$
1  randomly distribute items on the grid;
2  randomly place ants on the grid;
3  for $i := 1$ to maxIteration do
4      choose ant randomly;
5      move ant randomly by stepsize;
6      if antCarriesItem and cellIsEmpty then
7          $p_{drop}$ := calculate drop probability;
8          if randomDouble $\le p_{drop}$ then
9              drop item;
10         end
11     end
12     else if not (antCarriesItem) and not(cellIsEmpty) then
13         $p_{pick}$ := calculate pick probability;
14         if randomDouble $\le p_{pick}$ then
15             pick item;
16         end
17     end
18 end
```

Algorithm 3: Algorithm for ant-based clustering.

```
Input: number of clusters k, number of ants noAnts, maximum number of iterations maxIteration, pheromone decay parameter p, relative importance of the trail $\alpha$, relative importance of the visibility $\beta$, constant Q.
1  initialization;
2  lay equal pheromone on each path;
3  for $l := 1$ to maxIteration do
4      for $m := 1$ to noAnts do
5          while TWCV changed do
6              update pheromones $\tau_{ij}$;
7              assign each object to a cluster with probability $P^m$; $ij$
8              update cluster centers;
9              calculate TWCV;
10         end
11         if TWCV < bestTWCV then
12             bestTWCV:=TWCV;
13         end
14     end
15     perturbation;
16 end
```

Algorithm 4: Algorithm for ant K-means.

3.8. PSO.

In PSO [31], an individual grouped into a swarm is referred to as a particle. A swarm can be seen as a flock of birds flying toward an optimum. Therefore, a particle searches for the best solution, using the best position encountered by itself and its swarm. Thus, a wide area can be searched while heading toward an optimum. Each particle i represents a complete solution of the clustering problem and has to maintain the information of its current position x_i, its velocity v_i, and its personal best position y_i, where the personal best position is the position where the particle obtained the highest fitness value so far. In addition to the personal best position, there is also a global best position $\hat{y}$, determined from the entire swarm.

In each iteration t, the velocity and position of a particle are updated using the following formula [31]:

$$\begin{aligned} v_i(t+1) = w v_i(t) + c_1 r_1(t)\left(y_i(t) - x_i(t)\right) \\ + c_2 r_2(t)\left(y(\hat{t}) - x_i(t)\right), \end{aligned} \quad (13)$$

$$x_i(t+1) = x_i(t) + v_i(t+1), \quad (14)$$

where w is the inertia weight, c_1 and c_1 are constants, and r_1 and r_1 are random values between 0 and 1.

The particle's fitness at the current position is computed by means of a fitness function f encapsulating the characteristics of the optimization problem and thus reflecting the optimality of the solution. Then, the personal best position is updated by comparing the fitness value of the current position to the fitness value of the incumbent personal best position. If the fitness value of the current position is better, the incumbent personal best position is replaced by the current one (see Algorithm 5).

This algorithm is *mapped* to the peer clustering problem as follows. At the initialization step, the given number of clusters k is initialized by randomly assigning one peer to it, representing the cluster mean. Then, in each iteration, the solution of each particle has to be adapted. This means each peer of the solution is assigned to the best suiting (i.e., nearest) cluster. Afterward, the fitness of the current solution is calculated. This is done by using the ratio between the average intracluster distance (i.e., peer-to-cluster distance) and the average intercluster distance (i.e., cluster-to-cluster distance).

If the fitness of the current solution is smaller than the fitness of the personal and global best solution, respectively, the respective best solution is replaced by the current one. Subsequently, the centroids of each solution have to be updated. This complies with the abovementioned step of updating the velocity and position of a particle. In order to have a nonnumeric procedure which, at the same time, is in accordance with the formulas of equations (13) and (14), this is done the following way. The average of the cluster means of the current solution and the personal best solution is calculated, as well as of the cluster means of the current solution and the global best solution. Next, the two results of this operation are averaged again, resulting in the new cluster center. This way the new position of a particle is always influenced by the personal and global best solution, aiming at optimizing the current solution.

3.9. Slime Mold. The slime mold Dd is a social collective of amoebas, using spores to reproduce. The Dd's life cycle goes through five stages: vegetative movement, aggregation, mound, slug movement, and spore dispersal [32]. At the first stage, vegetative movement, amoebas are searching for food. An amoeba begins to starve if there is not enough food supply available. Thus, an amoeba remains in this state until the food supply diminishes. At this point, the aggregation stage begins, where amoebas show the first attempt of cooperative behavior. In this stage, there are two types of amoebas, namely, pacemakers and aggregating amoebas. A chemical called cAMP is emitted by a pacemaker. The aggregating amoebas aggregate toward pacemakers as they are attracted by cAMP. By this aggregation, amoebas are forming a mound. At this stage, the amoebas organize themselves into two groups, prestalk and prespore, based on their fitness level, i.e., how well-fed they are. The prestalk amoebas will be part of the fruiting body and, therefore, eventually die, whereas the prespore amoebas will be dispersed as spores. Together, the two groups form a slug, divided into a head, composed of prestalk amoebas, and a tail, formed by prespore amoebas. The slug moves into a direction where the possibility of a culmination, like a light source, exists. Then, at the last stage, it starts to form a fruiting body, where the prestalk amoebas become part of the fruiting body's stalk and will eventually die. The prespore amoebas are dispersed as spores, such that the Dd's life cycle can start again by entering the vegetative stage [32].

Reference [7] presents a swarm intelligent algorithm, which imitates the life cycle of the Dd slime mold. In this section, the slime mold algorithm for peer clustering is presented, which is based on the mentioned research [7].

At the first stage of the life cycle, the given number of amoebas *noAmoebae* is initialized. As every amoeba holds its own solution of the clustering problem, and this is done by randomly choosing k cluster means. In the second stage, aggregation, the clustering begins by assigning each peer to the nearest cluster center, representing the role of a pacemaker. Afterward, the cluster means are updated and the new TWCV is calculated. This is repeated until the old TWCV and the new TWCV converge to a limit given by ε. Then, in the mound stage, the one half of the amoebas having the better TWCV is categorized as prespore, and the other half is categorized as prestalk.

At the next stage, slug movement, the solution of each prespore amoeba is combined with the global best solution the same way as described in Algorithm 6, by averaging the two cluster means. At the dispersion stage, the global best solution is remembered by selecting the amoeba having the smallest TWCV. In the next iteration of the algorithm, the prespore amoebas are kept, whereas the prestalk amoebas are replaced by newly initialized amoebas.

The basic characteristics of the slime mold algorithm are presented in Table 2 [9].

3.10. Slime Mold K-Means. The slime mold K-means algorithm for clustering is based on the same biological approach as the pure slime mold for clustering. Therefore, the setting is basically the same, but the procedure differs from the original one by adding an additional K-means step, which is described in Algorithm 7. The mapping to the peer clustering problem of the slime mold K-means algorithm is very similar to the slime mold mapping. However, there are two deviations.

In the aggregation stage, each peer is assigned to the nearest cluster center, which represents the role of the pacemaker. After that, there is no more local optimization taking place.

In return, at the end of the dispersion stage, the solutions of all prespore amoebas are locally optimized by applying K-means.

The basic characteristics of the slime mold K-means algorithm are presented in Table 3 [9].

4. Evaluation

The proposed algorithms, slime mold for peer clustering and slime mold K-means, are benchmarked alongside ABC, ABC combined with K-means, ant-based clustering, ant K-means, fuzzy C-means, genetic K-means, hierarchical clustering, K-means, and PSO. The analysis encompasses the following stages: *) first, the parameter sensitivity analysis is done

```
Input: number of clusters $k$, number of particles noParticles, maximum number of iterations maxIteration, constant $w_1$, constant $w_2$
initialization;
for $j := 1$ to maxIteration do
    for $i := 1$ to noParticles do
        update velocity $v_i$;
        update position $x_i$;
        if $f(x_i) < f(y_i)$ then
            $y_i := x_i$
        end
        if $f(x_i) < f(\hat{y})$ then
            $\hat{y} := x_i$
        end
    end
end
```

Algorithm 5: Algorithm for PSO.

```
Input: number of clusters $k$, number of amoebae noAmoebae, maximum number of iterations maxIteration, restriction parameter $\varepsilon$
for $j := 1$ to maxIteration do
    Procedure vegetativeMovement():
        initialization of amoebae;
    end
    Procedure aggregation():
        for $i := 1$ to noAmoebae do
            while $|\text{TWCV} - \text{newTWCV}| > \varepsilon$ do
                assign each object to nearest cluster center;
                update cluster centers;
                calculate new TWCV;
            end
        end
    end
    Procedure mound():
        divide amoebae into prespore and prestalk;
    end
    Procedure slugMovement():
        for $i := 1$ to noPrespore do
            combine $\text{solution}_i$ with $\text{solution}_{\text{best}}$;
        end
    end
    Procedure dispersion():
        remember global best solution $\text{solution}_{\text{best}}$;
    end
end
```

Algorithm 6: Algorithm for slime mold.

Table 2: Basic characteristics of slime mold.

Mechanism of exploration	**Vegetative movement**
Mechanism of exploitation	Aggregation. Slug movement
Feasibility of solution	The feasible region is the space of all candidate solutions, i.e., the set of k nonempty clusters inhabited by all nodes in the considered network

```
Input: number of clusters $k$, number of amoebas $noAmoebae$, maximum number of iterations $maxIteration$
for $j := 1$ to maxIteration do
    Procedure vegetativeMovement():
        initialization of amoebas;
    end
    Procedure aggregation():
        for $i := 1$ to $noAmoebae$ do
            aggregate toward nearest pacemaker;
        end
    end
    Procedure mound():
        divide amoebas into prespore and prestalk;
    end
    Procedure slugMovement():
        for $i := 1$ to $noPrespore$ do
            combine $\text{solution}_i$ with $\text{solution}_{\text{best}}$;
        end
    end
    Procedure dispersion():
        remember global best solution $\text{solution}_{\text{best}}$;
        for $i := 1$ to $noPrespore$ do
        apply k-means;
        end
    end
end
```

Algorithm 7: Algorithm for slime mold K-means.

Table 3: Basic characteristics of slime mold K-means.

Mechanism of exploration	**Vegetative movement**
Mechanism of exploitation	Slug movement
Feasibility of solution	The feasible region is the space of all candidate solutions, i.e., the set of k nonempty clusters inhabited by all nodes in the considered network

because of many configurable parameters; *) second, the parameters which showed the best results are chosen for the comparative analysis; and *) third, a statistical significance of the obtained results is analyzed. Detailed information[7] about the experimental setup can be found in [33].

4.1. Metrics.

In order to proceed with the evaluation of abovementioned algorithms, metrics need to be defined. Therefore, in this subsection, the implemented metrics are described.

4.1.1. DBI.

The DBI [34] shall indicate the similarity of clusters. Therefore, it uses the following formula:

$$\text{DBI} = \frac{1}{n}\sum_{i=1}^{n} \max_{i\neq j}\left(\frac{S_i + S_j}{d\left(c_i, c_j\right)}\right), \tag{15}$$

where n is the number of clusters, Si is the average distance between the centroid of cluster i and all peers within cluster i, and $d(ci, cj)$ is the distance between the centroid of cluster i and cluster j. The DBI is nonnegative. The smaller the index is, the better the clustering result is.

4.1.2. DI.

The DI [12, 35] is a metric aiming at minimizing the intracluster distance, while having well-separated clusters (i.e., maximizing the intercluster distance). It uses the following formula:

$$\text{DI} = \frac{\min_{1\leq i<j\leq n}(d(i,j))}{\max_{1\leq k\leq n}(d'(x_k, y_k))}, \tag{16}$$

where $d(i, j)$ is the distance between the clusters i and j and $d'(x_k, y_k)$ is the distance of any pair of peers within cluster k. The higher the DI is, the better the result of the peer clustering is.

4.1.3. SC.

The SC [36] shall evaluate the validity of a clustering result and is often used to find the optimal number of clusters. It is calculated as follows:

$$a(i) = \frac{\sum d(i,j)}{|C_i| - 1}, \tag{17}$$

which is the average distance of peer i to all other peers within the same cluster C_i.

$$b(i) = \min_{C\neq C_i}(d(i,C)), \tag{18}$$

where $d(i,C)$ is the average distance between peer i to all peers of cluster C, for all clusters $C \neq C_i$. Then, those formulas are used to calculate the silhouette of i $s(i)$:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}. \quad (19)$$

The overall average silhouette width, here called silhouette coefficient, $s(k)$ is used to measure the validity of the clustering result:

$$s(k) = \frac{\sum_{i=1}^{n} s(i)}{n}, \quad (20)$$

where n is the number of peers which were clustered.

The SC ranges between −1 and 1. The higher the coefficient, the better the peer clustering result.

4.1.4. ADC. The ADC is a simple metric to evaluate a peer clustering result and is developed in the course of this research. The calculation works as follows.

First, the average distance of a peer i to all other peers in the same cluster C_x is calculated:

$$a(i, C_x) = \frac{\sum_{j=1}^{|C_x|} d(i,j)}{|C_x| - 1}. \quad (21)$$

This is done for every peer in the cluster C_x. Then, the average cluster within distance for cluster C_x is calculated by summing up all object distances and dividing them through the number of cluster members:

$$b(C_x) = \frac{\sum_{i=1}^{|C_x|} a(i, C_x)}{|C_x|}. \quad (22)$$

After that, the ADC is calculated by

$$\mathrm{ADC} = \frac{\sum_{x=1}^{k} b(C_x)}{k}, \quad (23)$$

where k is the number of clusters. Obviously, the smaller the ADC is, the better the clustering result is.

4.2. Benchmark Methodology. All benchmarks are carried out on Google Cloud Platform's Compute Engine[8]. In particular, a "n1-standard-16" instance was used, which includes 16 vCPUs and 60 GB RAM. The following processors are available for n1 machine types: 2.0 GHz Intel Xeon Scalable Processor, 2.2 GHz Intel Xeon E5 v4, 2.3 GHz Intel Xeon E5 v3, 2.5 GHz Intel Xeon E5 v2, and 2.6 GHz Intel Xeon E5.

The general benchmark setups are the following:

1. **P2P network nodes:** The amount of the nodes is varied in three levels: low (L_n), medium (M_n), and high (H_n), where $L_n = 50$, $M_n = 100$, and $H_n = 200$. This means, for example, that in a network of medium size, 100 storage peers, 100 worker peers, and 100 client peers are present. Furthermore, it shall be noted that no churn is simulated. Thus, there is no fluctuation of node participation.
2. **Requests:** For the sensitivity analysis, the amount of requests per client peer is varied in three levels: low (L_r), medium (M_r), and high (H_r), where $L_r = 1$, $M_r = 5$, and $H_r = 10$ query and job requests per client peer. On the contrary, all competitive benchmarks are executed with exactly three queries and job requests per client peer, as the requests have shown to have no direct impact on the performance of the peer clustering algorithms.
3. **Benchmark execution:** Each configuration of an algorithm is executed 100 times, in order to have a variety of different node initializations. Thus, in the result data, the average metric values are recorded.

The suggested default values are used for the rest of the framework parameters.

The metrics used for the evaluation are execution time, DBI, DI, SC, and ADC. The metric of execution time measures the performance of the algorithms which can be important, especially in time-critical scenarios. In order to examine the quality of the peer clustering solutions from different perspectives, three established and well-known metrics are chosen. While the DBI [34] uses artificial cluster means to calculate the ratio of the within cluster cohesion to the between cluster separation, the other two measures use only the distances between individual nodes. Furthermore, the DI [12, 35] uses the worst case scenarios, i.e., the global minimum distance between two clusters and the global maximum distance between any pair of peers within the same cluster, to calculate the ratio between cohesion and separation, whereas the DBI [34] and the SC [37] are calculated with average values. In contrast to the other two metrics which directly calculate the ratio between cohesion and separation, the SC normalizes the result by subtracting the average within-cluster distance from the minimum between-cluster distance and dividing the result by the greater of both numbers. Additionally, one simple metric, ADC, is introduced. It mainly distinguishes from the other metrics as it only focuses on the cohesion within the clusters, while neglecting the well-separation of the clusters.

4.3. Implementation Platform. A framework based on the Peer Model [38] is implemented for benchmarking and comparison of clustering algorithms for unstructured P2P networks. It is domain independent, supports arbitrary configurations, and serves only as a necessary basement for the used algorithms. The framework allows the plugging of different peer clustering algorithms in order to allow an easy exchangeability of the applied algorithms and systematic benchmarking as well as comparison of these algorithms. It is used to find the best suiting algorithm for a specific problem. The main characteristics of underlying framework are only very briefly mentioned without detailed description because the main focus in this paper concerns the peer clustering algorithm. The Peer Model is a space-based coordination middleware for distributed environments with a data-driven workflow. The strict separation of the coordination from the application logic allows reusing proven coordination patterns. The main component of the Peer Model is the peer, which represents a structured, reusable, and addressable component. More details of the Peer Model can be found in [39, 40].

4.4. Dataset Details. In this subsection, the dataset details are provided including the sensitivity analysis based on these datasets. The goal of the sensitivity analysis is to find the best combination of parameters for each benchmarked algorithm. Each test case is executed 100 times (due to the nondeterministic nature of the benchmarked algorithms) and the recorded results are averaged and stored for future analysis. A selection of the best set of parameters' values is a computationally expensive and delicate task. In order to perform an automatic parameter tuning, racing [41] has been chosen. This method focuses only on well performing configurations and discards those ones that do not perform well enough. It starts by running new configurations against a small subset of the testing instances. Only those configurations, which do not perform significantly worse than the best yet found configuration, are chosen for the subsequent runs. The number of testing instances is increased in each iteration until only the best configuration is left or a maximum number of test runs is reached. In order to effectively compare the candidate configurations, the pairwise t-test is used.

Therefore, regarding sensitivity analysis, either recommended values from the literature (cited in respective Tables 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, and 25) are used or in case of preliminary testing, a thorough investigation is used by means of automatic parameter tuning. This parameter tuning is performed on the parameter's value range recommended in the previous research work or determined in preliminary benchmarks. Regarding number of nodes in network and the number of clusters, a selection of parameter values is done in order to retain scalability.

As for the most benchmarked algorithms, the number of clusters has to be parameterized, and a global range for this value is used, depending on the number of nodes in the network, i.e., number of clusters = $\{n*0.1;\ n*0.2;\ n*0.3;\ n*0.4;\ n*0.5\}$, where n is the number of nodes in the network. These values are used for the following reasons: As it makes a great difference for the quality of the clustering solution whether 10 or 100 peers are clustered into 3 clusters, the number of clusters shall be dependent on the number of nodes. Furthermore, the upper limit for k is set to $n/2$, as otherwise a balanced distribution would be harder to accomplish. As the number of clusters is required in nearly each benchmarked algorithm, it can be considered to be one of the most important parameters and therefore five values are tested, fairly distributed between 10% and 50% of the number of nodes to be clustered.

The number of iterations *maxIteration* and the number of agents, like *noAnts* and *noAmoebae*, are also parameters occurring more often; therefore, for them, three global values are specified to be tested, in case there does not exist an expert recommendation. For the number of agents, the

Table 4: ABC parameter values before sensitivity analysis.

Parameter	Value (range)	Source
k	$n*0.1$, $n*0.2$, $n*0.3$, $n*0.4$, $n*0.5$	Preliminary benchmarks
noSources	20	[27]
maxIteration	1000	[27]

Table 5: ABC sensitivity analysis results.

Nodes n	Requests	k
50	1	$n*0.5$
50	5	
50	10	
100	1	
100	5	
100	10	

Table 6: ABCK parameter values before sensitivity analysis.

Parameter	Value (range)	Source
k	$n*0.1$, $n*0.2$, $n*0.3$, $n*0.4$, $n*0.5$	Preliminary benchmarks
noSources	10	[28]
maxIteration	20	[28]

Table 7: ABCK sensitivity analysis results.

Nodes n	Requests	k
50	1	$n*0.4$
50	5	
50	10	
100	1	
100	5	
100	10	

Table 8: Ant-based clustering parameter values before sensitivity analysis.

Parameter	Value (range)	Source
Stepsize	1	[29]
maxIteration	$\max(1{,}000{,}000;\ \sqrt{2000*n})$	[42]
Kpick	0.1	[43]
Kdrop	0.1	[43]
α	0.3, 0.6, 0.9	Preliminary benchmarks

Table 9: Ant-based clustering sensitivity analysis results.

Nodes n	Requests	α
50	1	0.3
50	5	
50	10	
100	1	
100	5	
100	10	

Table 10: Ant K-means parameter values before sensitivity analysis.

Parameter	Value (range)	Source
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	Preliminary benchmarks
noAnts	10, 30, 50	Preliminary benchmarks
maxIteration	20, 50, 100	Preliminary benchmarks
p	0.9	[29]
α	0.5	[29]
β	1	[29]
Q	1	[29]

Table 11: Ant K-means sensitivity analysis results.

Nodes n	Requests	k	noAnts	maxIteration
50	1	$n * 0.1$	10	20
50	5			
50	10			
100	1			
100	5			
100	10			

Table 12: Fuzzy C-means parameter values before sensitivity analysis.

Parameter	Value (range)	Source
c	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	Preliminary benchmarks
m	1.5, 2.2, 3.0	[11]
acceptanceBorder	$\underline{1}\ c$	[44]

Table 13: Fuzzy C-means sensitivity analysis results.

Nodes n	Requests	c	m
50	1	$n * 0.1$	1.5
50	5		
50	10		
100	1		
100	5		
100	10		

Table 14: Genetic K-means parameter values before sensitivity analysis.

Parameter	Value (range)	Source
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	Preliminary benchmarks
maxGeneration	100	[10]
nrOf chromosomes	50	[10]
c	2	[10]
alleleModifier	1, 5, 10	Preliminary benchmarks
mutationProbability	0.05	[10]

Table 15: Genetic K-means sensitivity analysis results.

Nodes n	Requests	k	alleleModifier
50	1	$n * 0.1$	10
50	5		
50	10		
100	1		
100	5		
100	10		

Table 16: Hierarchical clustering parameter values before sensitivity analysis.

Parameter	Value (range)	Source
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	Preliminary benchmarks

Table 17: Hierarchical clustering sensitivity analysis results.

Nodes n	Requests	k
50	1	$n * 0.5$
50	5	
50	10	
100	1	
100	5	
100	10	

Table 18: K-means parameter values before sensitivity analysis.

Parameter	Value (range)	Source
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	Preliminary benchmarks

Table 19: K-means sensitivity analysis results.

Nodes n	Requests	k
50	1	$n * 0.5$
50	5	
50	10	
100	1	
100	5	
100	10	

Table 20: PSO parameter values before sensitivity analysis.

Parameter	Value (range)	Source
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	Preliminary benchmarks
noParticles	10	[31]
maxIteration	100	[31]
w_1	0.5	[31]
w_2	0.5	[31]

minimum and maximum recommended values from other algorithms are used as lower and upper limit (10 from PSO and 50 from genetic K-means), whereas the middle value is the average. Thus, the number of agents is tested for the values 10, 30, and 50. For the number of iterations, the lower limit is also defined by the minimum recommended value (20 from ABCK), whereas the upper limit is set to 100 as it is a value twice recommended (from genetic K-means and PSO) and due to performance reasons. The middle value is

Table 21: PSO sensitivity analysis results.

Nodes n	Requests	k
50	1	
50	5	
50	10	
100	1	$n * 0.5$
100	5	
100	10	

Table 22: Slime mold parameter values before sensitivity analysis.

Parameter	Value (range)	Source
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * .4$, $n * 0.5$	Preliminary benchmarks
noAmoebae	10, 30, 50	Preliminary benchmarks
maxIteration	20, 50, 100	Preliminary benchmarks
ε	0.3, 0.6, 0.9	Preliminary benchmarks

Table 23: Slime mold sensitivity analysis results.

Nodes n	Requests	k	noAmoebae	maxIteration	ε
50	1				
50	5				
50	10	$n * 0.5$	10	20	0.6
100	1				
100	5				
100	10				

Table 24: Slime mold K-means parameter values before sensitivity analysis.

Parameter	Value (range)	Source
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	Preliminary benchmarks
noAmoebae	10, 30, 50	Preliminary benchmarks
maxIteration	20, 50, 100	Preliminary benchmarks

Table 25: Slime mold K-means sensitivity analysis results.

Nodes n	Requests	k	noAmoebae	maxIteration
50	1			
50	5			
50	10	$n * 0.5$	10	20
100	1			
100	5			
100	10			

set to 50. Thus, *maxIteration* is tested for the values 20, 50, and 100. Also, other nonrecommended parameters are tested with three values. In detail, for all peer clustering algorithm parameters being subject of the sensitivity analysis, the optimal parameter value is determined based on the metrics mentioned above for all combinations of network sizes (L_n, M_n) and request levels (L_r, M_r, H_r). These optimal values are then used in the competitive benchmarks.

4.4.1. *ABC.*

As many other algorithms, the ABC peer clustering algorithm requires a given number of clusters k, in which to cluster the nodes. Therefore, as mentioned before, a global range is used which depends on the number of nodes n participating in the network, i.e., the number of objects to be clustered. Therefore, the value of k ranges between $n/10$ and $n/2$. The values of all other parameters used for the competitive benchmark are based on the recommendations in [27]. The parameter values and ranges, respectively, are shown in Table 4, whereas the results of the sensitivity analysis are depicted in Table 5. As can be seen, for each combination of network sizes and request levels, the value of k performed best with $k = n * 0.5$. Therefore, in a network consisting of 100 nodes, $k = 50$.

4.4.2. *ABCK.*

Also for the peer clustering algorithm ABCK, the number of clusters k has to be determined. The other parameter values of the algorithm are recommended in [28].

The parameter values and value ranges are listed in Table 6.

In contrast to ABC, the ABCK algorithm performs best with a value of $k = n * 0.4$ for each combination setup, which is stated in Table 7. It may be noticeable that the values for *noSources* and *maxIteration* differ for ABC and ABCK, although the algorithms are rather similar. This is because it was decided to use recommended values for each algorithm if available.

4.4.3. *Ant-Based Clustering.*

The ant-based clustering algorithm is the only benchmarked algorithm which does not require a parameterized number of clusters. The values for *maxIteration*, k_{pick}, k_{drop} and *stepsize* are suggested in [29, 42, 43], respectively. The only parameter to be determined through sensitivity analysis is the scaling parameter α. α ranged between 0 and 1 and was therefore tested with the values given below.

The parameter values and value ranges before the sensitivity analysis, respectively, can be seen in Table 8, whereas its results are shown in Table 9.

As presented in Table 9, for each combination of network sizes and request level, the value of $\alpha = 0.3$ outperformed the other inspected values for the scaling parameter α.

4.4.4. *Ant K-Means.*

The ant K-means algorithm has three parameter values to be determined, namely, k, *noAnts*, and *maxIteration*. The values for the parameters p, α, β, and Q are recommended in [30]. The parameter values and value ranges are shown in Table 10.

The results of the sensitivity analysis can be seen in Table 11. It illustrates that the values $k = n * 0.1$, *noAnts* = 10, and *maxIteration* = 20 performed best across all combination setups.

4.4.5. *Fuzzy C-Means.*

For the fuzzy C-means algorithm, the number of clusters c has to be determined. According to [44], a fuzzifier m chosen between 1.5 and 3.0 gives good results for most data. The value for the *acceptanceBorder* is suggested in [44]. The parameter values and value ranges, respectively, can be seen in Table 12, whereas the sensitivity analysis results are shown in Table 13. As illustrated in Table 13, for each combination of network sizes and request levels, the values of $c = n * 0.1$ and $m = 1.5$ performed best.

4.4.6. Genetic K-Means. Most of its parameter values are recommended in [10]. Only the values of the number of clusters k and the *alleleModifier* have to be determined by the sensitivity analysis. According to [10], *alleleModifier* is 1; therefore, its lower limit is 1, whereas the upper limit is set to 10 due to preliminary benchmarks. Then, the natural number 5 is chosen for the middle value, as it is about the mean. Table 14 shows the values and value ranges, respectively, while the results of the sensitivity analysis can be seen in Table 15.

The genetic K-means algorithm performs best with a value of $k = n * 0.1$ and $alleleModifier = 10$ for each combination setup. This indicates that the algorithm performs better when the probability for mutation is higher.

4.4.7. Hierarchical Clustering. For the hierarchical clustering algorithm, the number of cluster k has to be determined. The parameter value range is shown in Table 16, whereas its results can be inspected in Table 17.

Across all combination setups, the hierarchical clustering algorithm performs best with a value of $k = n * 0.5$, shown in Table 17.

4.4.8. K-Means. The K-means algorithm only has the number of clusters parameter k to be determined by the sensitivity analysis. Therefore, Table 18 lists the value range, whereas Table 19 shows the sensitivity analysis results.

The sensitivity analysis shows that K-means also performs best with a parameter value of $k = n * 0.5$.

4.4.9. PSO. For the PSO peer clustering algorithm, only the number of clusters parameter k has to be determined, as all other parameter values used for the competitive benchmark are recommended in [31]. The parameter value and value ranges before the sensitivity analysis, respectively, are listed in Table 20, while its results are presented in Table 21.

As can be seen in Table 20, for each combination of network sizes and request levels, the PSO algorithm's value of k performs best with $k = n * 0.5$.

4.4.10. Slime Mold. The slime mold peer clustering algorithm has multiple parameter values to be investigated by the sensitivity analysis. Those parameter value ranges are presented in Table 22. The threshold value ε has to be carefully chosen, as it is important for the local optimization. If the lower limit is chosen too small, the algorithm could be stuck at local optimization at the expense of performance; therefore, it is set to 0.3. If the upper limit is chosen too big, the local optimization ends too early which may affect the quality of the clustering solution negatively. Therefore, the upper limit is set to 0.9, whereas the mean of 0.6 is taken for the middle value.

The results of the sensitivity analysis can be seen in Table 23. Also, the slime mold algorithm performs best with the same values for all combination of network sizes and request levels, i.e., $k = n * 0.5$, $noAmoebae = 10$, $maxIteration = 20$, and $\varepsilon = 0.6$.

4.4.11. Slime Mold K-Means. Just like the slime mold algorithm, all of the slime mold K-means algorithm's parameter values have to be determined. The parameter values and value ranges are listed in Table 24.

The results of the sensitivity analysis can be seen in Table 25. It shows that the values $k = n * 0.5$, $noAmoebae = 10$, and $maxIteration = 20$ performed best across all combinations of network size and request level.

4.5. Simulation Results. This subsection provides simulation results graphically presented and based on the raw data results (see Appendix, Tables A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, and A11).

In Figure 3, the *execution time* for all benchmarked algorithms is depicted. Figure 4 provides a closer look into these algorithms.

Figure 5 presents the overall results for the DBI for all benchmarked algorithms. In Figure 6, the comparison of the DBI results between worker and storage peer clustering is depicted for ABC, ABCK, ant-based clustering, and ant K-means, whereas in Figure 7, it is shown for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means.

In Figure 8, the DI results for all benchmarked algorithms are depicted. Figure 9 provides the comparison of the DI results between worker peer clustering and storage peer clustering for ABC, ABCK, ant-based clustering, ant K-means, and fuzzy C-means, while Figure 10 provides it for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means.

Figure 11 presents the overall results for the SC for all benchmarked algorithms. In Figure 12, the comparison of the SC results between worker and storage peer clustering is depicted for ABC, ABCK, ant-based clustering, ant K-means, and fuzzy C-means, whereas in Figure 13, it is shown for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means.

In Figure 14, the ADC results for all benchmarked algorithms are depicted. Figure 15 provides the comparison of the ADC results between worker peer clustering and storage peer clustering for ABC, ABCK, ant-based clustering, ant K-means, and fuzzy C-means, while Figure 16 provides it for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means.

4.6. Discussion and Evaluation. In Figure 1, the *execution time* for all benchmarked algorithms is depicted. Figure 2 provides a closer look into these algorithms. Those five algorithms, fuzzy C-means, ant-based clustering,

Table A1: ABC raw result data.

Type	Nodes	Time (s)	DBI	DI	SC	ADC
Worker	50	25.20	1.5053	0.2619	0.0662	0.6160
	100	85.14	1.7040	0.2287	0.0037	0.7146
	200	278.20	1.6696	0.2247	−0.0458	0.8456
Storage	50	25.77	1.4112	0.4041	0.0429	0.5350
	100	87.53	1.5507	0.3165	−0.0261	0.6021
	200	263.67	1.6591	0.4167	−0.1685	0.9457
All	50	25.48	1.4583	0.3330	0.0545	0.5755
	100	86.34	1.6274	0.2726	−0.0112	0.6583
	200	270.93	1.6643	0.3207	−0.1072	0.8957

Table A2: ABCK raw result data.

Type	Nodes	Time (s)	DBI	DI	SC	ADC
Worker	50	2.10	1.1258	0.3534	0.2230	0.6745
	100	8.27	1.1541	0.3273	0.2102	0.6581
	200	37.53	1.1948	0.3233	0.1979	0.6403
Storage	50	1.87	1.0744	0.6046	0.3326	0.4196
	100	7.81	1.1352	0.5528	0.3207	0.4499
	200	36.61	1.2013	0.5086	0.2908	0.4471
All	50	1.98	1.1001	0.4790	0.2778	0.5470
	100	8.04	1.1447	0.4400	0.2654	0.5540
	200	37.07	1.1980	0.4160	0.2443	0.5396

Table A3: Ant-based clustering raw result data.

Type	Nodes	Time (s)	DBI	DI	SC	ADC
Worker	50	0.51	2.2532	0.3566	−0.0826	1.2113
	100	0.55	2.3191	0.3311	−0.1141	1.2045
	200	0.57	2.3400	0.3061	−0.1303	1.1831
Storage	50	1.17	1.9799	0.5667	−0.0443	0.9695
	100	2.01	2.0073	0.7000	−0.0640	0.9593
	200	2.11	2.1104	0.6000	−0.0655	0.9437
All	50	0.84	2.1165	0.4617	−0.0634	1.0986
	100	0.79	2.1632	0.5156	−0.0891	1.1002
	200	0.83	2.2252	0.4531	−0.0979	1.0802

Table A4: Ant K-means raw result data.

Type	Nodes	Time	DBI	DI	SC	ADC
Worker	50	6.18	1.5582	0.4922	0.0711	1.0180
	100	66.94	1.6239	0.4426	0.0512	1.0069
	200	524.95	1.7113	0.5143	0.0324	0.9760
Storage	50	5.19	1.5636	1.0000	0.0798	0.8721
	100	42.82	1.6277	0.9539	0.0704	0.8941
	200	482.40	1.6542	0.9333	0.0567	0.9176
All	50	5.68	1.5609	0.7461	0.0755	0.9472
	100	54.88	1.6258	0.6982	0.0608	0.9531
	200	503.68	1.6828	0.7238	0.0445	0.9407

hierarchical clustering, slime mold and slime mold K-means, are quite close to each other. The ant-based clustering algorithm shows a very constant execution time for each network size level. This effect is due to the fact that it is the only benchmarked algorithm which does not depend on the number of peers to be clustered. Its complexity is only dependent on the number of iterations and the number of ant agents moving across the grid. The ant-based clustering algorithm is also the only one where the execution time of worker peer clustering and storage peer clustering differs notably, as storage peer clustering takes about twice as long as worker peer clustering.

Table A5: Fuzzy C-means raw result data.

Type	Nodes	Time	DBI	DI	SC	ADC
	50	0.02	—	0.0182	−0.0789	1.1150
Worker	100	0.09	—	0.0041	−0.0462	1.1076
	200	0.79	—	0.0015	0.0135	1.0900
	50	0.09	—	0.7500	0.000	0.9696
Storage	100	0.29	—	0.5333	0.000	0.9877
	200	1.68	—	0.5667	0.000	0.9940
	50	0.05	—	0.3841	−0.0395	1.0423
All	100	0.19	—	0.2687	−0.0231	1.0476
	200	1.23	—	0.2841	0.0068	1.0420

Table A6: Genetic K-means raw result data.

Type	Nodes	Time (s)	DBI	DI	SC	ADC
	50	10.38	1.5802	0.3919	0.0773	0.8903
Worker	100	35.31	1.5587	0.3918	0.0767	0.8950
	200	138.773	1.5644	0.3876	0.0707	0.8877
	50	10.14	1.6621	0.7243	0.1535	0.7294
Storage	100	38.46	1.6271	0.7113	0.1576	0.7627
	200	139.21	1.6046	0.7124	0.1436	0.7441
	50	10.26	1.6212	0.5581	0.1154	0.8099
All	100	36.89	1.5929	0.5515	0.1172	0.8289
	200	138.99	1.5845	0.5500	0.1071	0.8159

Table A7: Hierarchical clustering raw result data.

Type	Nodes	Time (s)	DBI	DI	SC	ADC
	50	0.14	0.8281	0.3958	0.4112	0.3437
Worker	100	0.67	0.8781	0.3698	0.4158	0.3194
	200	3.86	0.9201	0.3490	0.4166	0.3359
	50	0.14	0.8965	0.6594	0.4558	0.2547
Storage	100	0.70	0.9485	0.6155	0.4557	0.2835
	200	4.22	0.9489	0.5729	0.4294	0.3858
	50	0.14	0.8620	0.5231	0.4335	0.2992
All	100	0.68	0.9133	0.4927	0.4358	0.3015
	200	4.04	0.9345	0.4610	0.4230	0.3108

Table A8: K-means raw result data.

Type	Nodes	Time (s)	DBI	DI	SC	ADC
	50	0.01	1.0458	0.2785	0.2777	0.5110
Worker	100	0.03	1.0726	0.2300	0.2770	0.5056
	200	27.02	1.1965	0.3663	0.2494	0.4940
	50	0.01	0.9889	0.4886	0.3791	0.3080
Storage	100	0.04	1.0378	0.4637	0.3825	0.3363
	200	27.56	1.2395	0.6393	0.3107	0.3279
	50	0.01	1.0173	0.3835	0.3284	0.4095
All	100	0.04	1.0552	0.3469	0.3298	0.4210
	200	27.29	1.2180	0.5028	0.2801	0.4109

The slime mold algorithm and the slime mold K-means algorithm show a good execution time in general and specifically compared to other swarm intelligent algorithms. They start with a comparable execution time at network size level L_n, and slime mold is 21% faster at level M_n and 30% faster at level H_n. Thus, the execution time of slime mold increases by 3.7 times (L_n to M_n) and by 3.9 times (M_n to H_n), while the execution time of slime mold K-means increases by a factor of 4.2 (L_n to M_n) and by a factor of 4.4 (M_n to H_n). Thus, the slime mold algorithm outperforms all other algorithms in terms of scalability related to execution time. Interestingly, while K-means algorithm scales rather good from network size

Table A9: PSO raw result data.

Type	Nodes	Time (s)	DBI	DI	SC	ADC
Worker	50	4.07	0.9781	0.2625	0.2792	0.4929
	100	17.34	1.0523	0.1916	0.2670	0.5047
	200	70.54	1.1410	0.1282	0.2559	0.5131
Storage	50	4.00	0.9397	0.4874	0.3780	0.3060
	100	17.29	1.0413	0.4699	0.3766	0.3284
	200	72.77	1.1066	0.4671	0.3580	0.3067
All	50	4.04	0.9589	0.3749	0.3286	0.3994
	100	17.31	1.0468	0.3307	0.3218	0.4165
	200	71.43	1.1238	0.2977	0.3069	0.4099

Table A10: Slime mold raw result data.

Type	Nodes	Time (s)	DBI	DI	SC	ADC
Worker	50	0.79	1.2406	0.3168	0.1701	0.6459
	100	2.93	1.2816	0.3035	0.1596	0.6467
	200	11.39	1.3102	0.2678	0.1599	0.6403
Storage	50	0.81	1.2235	0.5935	0.2416	0.5136
	100	2.97	1.2519	0.5462	0.2379	0.5450
	200	11.55	1.2716	0.4995	0.2190	0.5215
All	50	0.80	1.2320	0.4552	0.2058	0.5798
	100	2.95	1.2667	0.4249	0.1988	0.5959
	200	11.47	1.2909	0.3837	0.1895	0.5809

Table A11: Slime mold K-means raw result data.

Type	Nodes	Time (s)	DBI	DI	SC	ADC
Worker	50	0.92	1.2091	0.3210	0.1933	0.6436
	100	3.88	1.2328	0.2965	0.1893	0.6362
	200	16.67	1.2422	0.2545	0.2067	0.6155
Storage	50	0.87	1.1828	0.6008	0.2678	0.5047
	100	3.61	1.1815	0.5531	0.2792	0.5258
	200	16.01	1.2160	0.5186	0.2487	0.5216
All	50	0.89	1.1960	0.4609	0.2305	0.5742
	100	3.75	1.2071	0.4248	0.2342	0.5810
	200	16.34	1.2991	0.3865	0.2277	0.5685

level L_n to M_n, it then shows an increase by a factor of about 682 to level H_n, which may be due to the fact that the algorithm is repeated until no more cluster changes occur. Consequently, six algorithms require more than 20 s for their execution, first and foremost ant K-means with an average execution time of 503.68 s for a network size of level H_n. Thus, in terms of execution time, ant K-means algorithm scales definitely worst with a time increase of about 10 times each network size level, as it is also repeated until no peer changes its cluster anymore.

As execution time is an important factor for clustering algorithms, but by far not the only one, four metrics measuring the quality of the peer clustering result are now discussed.

Figure 3 presents the overall results for the DBI for all benchmarked algorithms. In Figure 4, the comparison of the DBI results between worker and storage peer clustering is depicted for ABC, ABCK, ant-based clustering, and ant K-means, whereas in Figure 5, it is shown for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means.

The smaller the DBI is, the better the clustering result is. The hierarchical clustering algorithm includes a low increase rate of 6% (L_n to M_n) and 2% (M_n to H_n). It is followed by PSO, K-means, and ABCK, where PSO and K-means scale rather poorly in comparison. While ABCK increases by 4% (L_n to M_n) and 5% (M_n to H_n), PSO increases by 9% and 7% per network size level and K-means even by 4% and 15%. This may be due to the fact that in hierarchical clustering, in contrast to the other algorithms, no random cluster initialization is made, but instead the two globally best matching clusters are merged together, starting with each peer having its own cluster. Slime mold K-means and slime mold follow immediately, scaling fairly well with an increase of 1% (L_n to M_n) and 2% (M_n to H_n), and 3% and 2%, respectively. Slime mold and slime mold K-means impress in terms of scalability regarding effectiveness. The algorithm which definitely performed worst regarding the DBI is

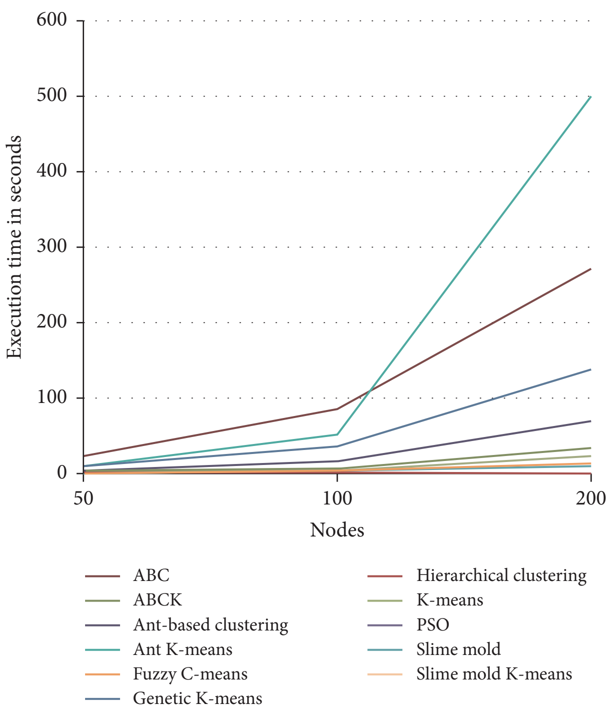

Figure 3: Execution time results.

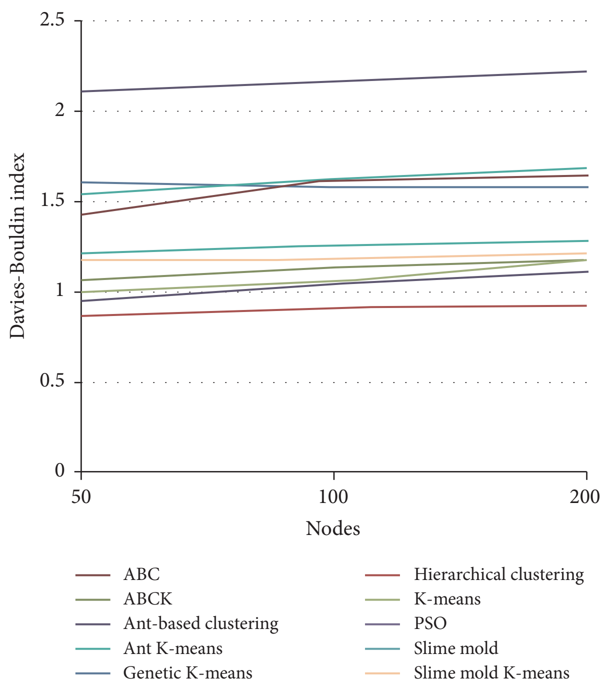

Figure 5: Davies–Bouldin results for all network size levels.

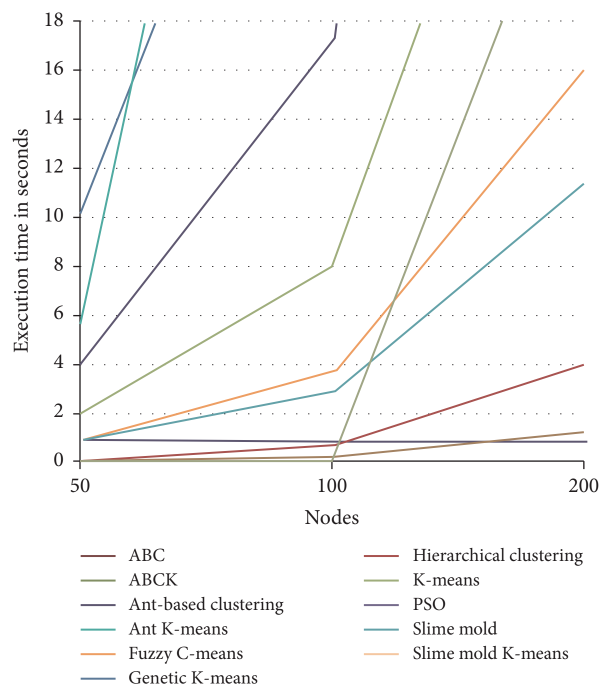

Figure 4: Execution time results zoomed in.

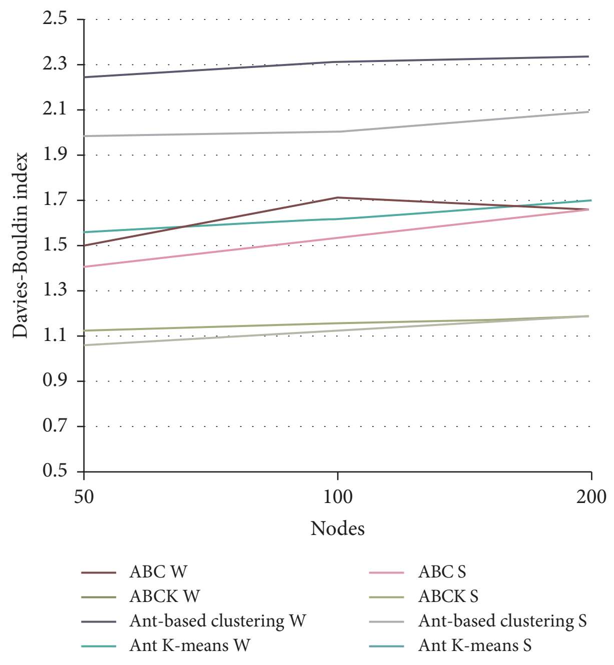

Figure 6: Comparison of Davies–Bouldin index results between worker and storage peers for ABC, ABCK, ant-based clustering, and ant K-means for all network size levels.

the ant-based clustering algorithm. Also, it is the algorithm which shows the most discrepancy between worker peer clustering and storage peer clustering, with storage peer clustering being 12% (50 nodes), 13% (100 nodes), and 10% (200 nodes) more effective than worker peer clustering. This could be caused by the randomness of how the peers are distributed on the grid on which the ants move around. Overall, genetic K-means and hierarchical clustering are the only benchmarked algorithms which show better results for worker peer clustering than for storage peer clustering. Furthermore, K-means algorithm performs better in storage peer clustering on network size levels L_n and M_n but shows better results for worker peer clustering on a network consisting of 200 nodes.

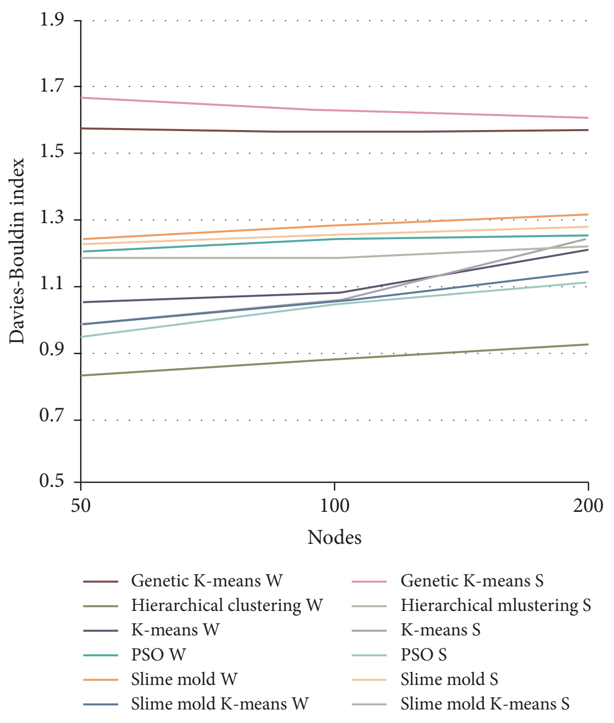

FIGURE 7: Comparison of Davies–Bouldin index results between worker and storage peers for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means for all network size levels.

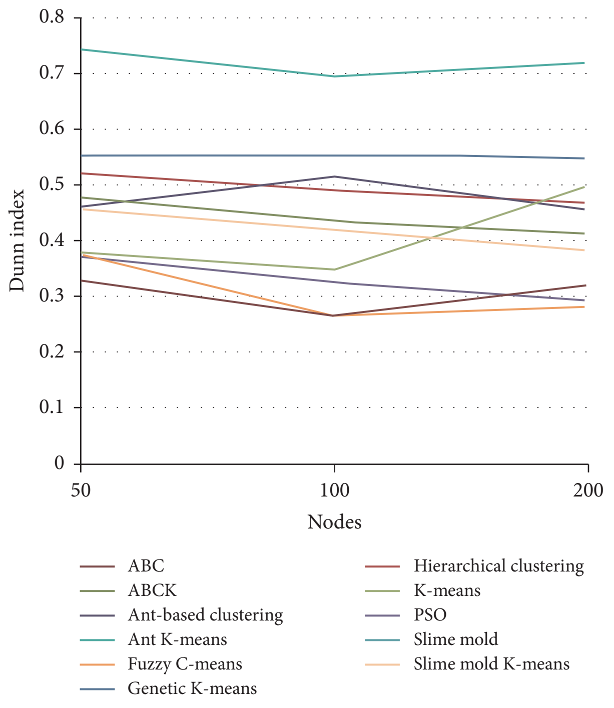

FIGURE 8: Dunn index results for all network size levels.

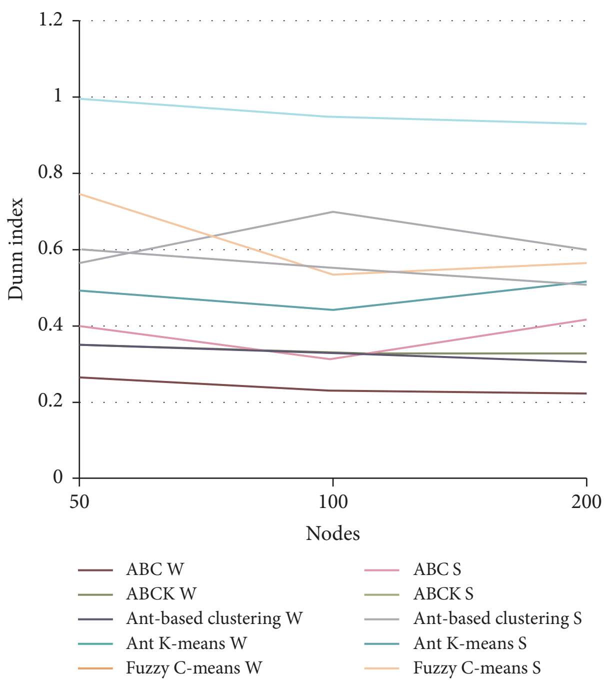

FIGURE 9: Comparison of Dunn index results between worker and storage peers for ABC, ABCK, ant-based clustering, ant K-means, and fuzzy C-means for all network size levels.

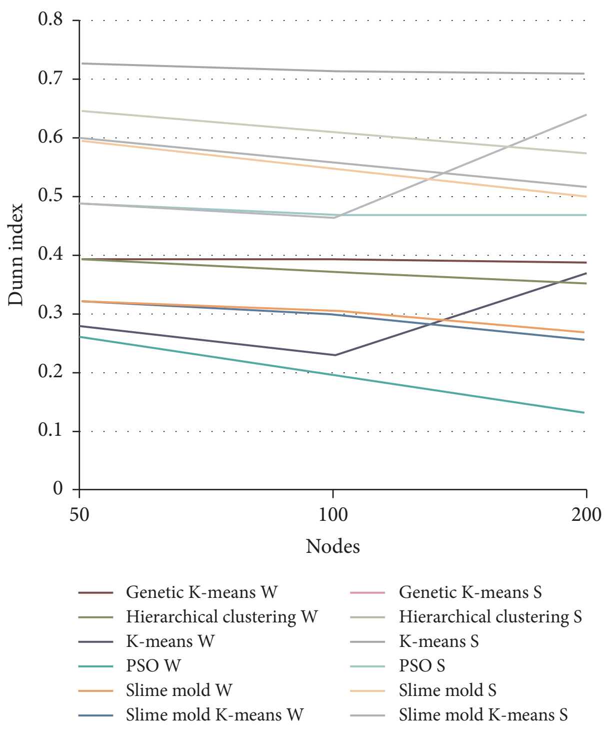

FIGURE 10: Comparison of Dunn index results between worker and storage peers for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means for all network size levels.

In Figure 6, the DI results for all benchmarked algorithms are depicted. Figure 7 provides the comparison of the DI results between worker peer clustering and storage peer clustering for ABC, ABCK, ant-based clustering, ant K-means, and fuzzy C-means, while Figure 8 provides it for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means.

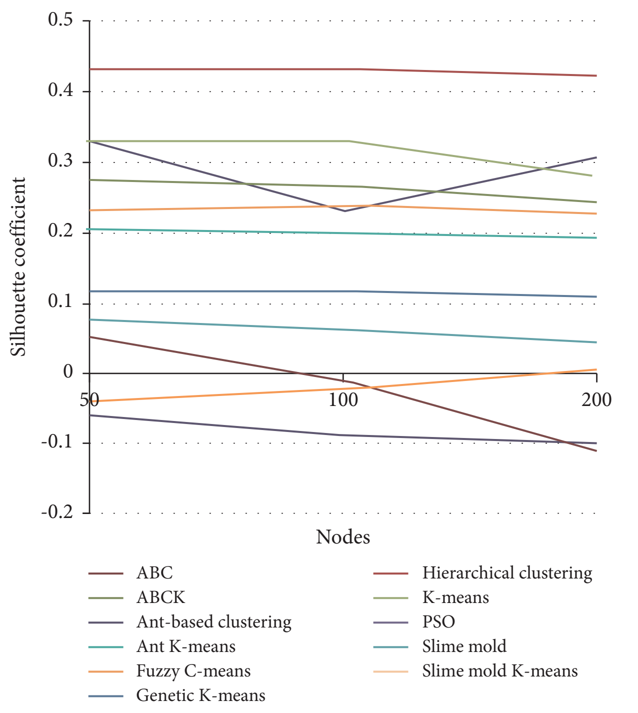

Figure 11: Silhouette coefficient results for all network size levels.

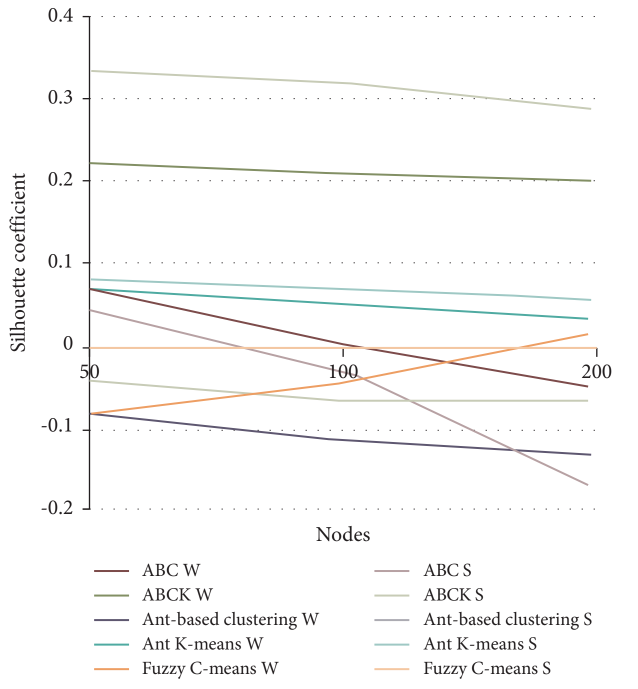

Figure 12: Comparison of silhouette coefficient results between worker and storage peers for ABC, ABCK, ant-based clustering, ant K-means, and fuzzy C-means for all network size levels.

The higher the DI is, the better the clustering result is. By far, the ant K-means algorithm shows the best results for the average DI, as it is 34% (50 nodes), 27% (100 nodes), and 32% (200 nodes) more effective than its successor genetic K-means, indicating that ant K-means

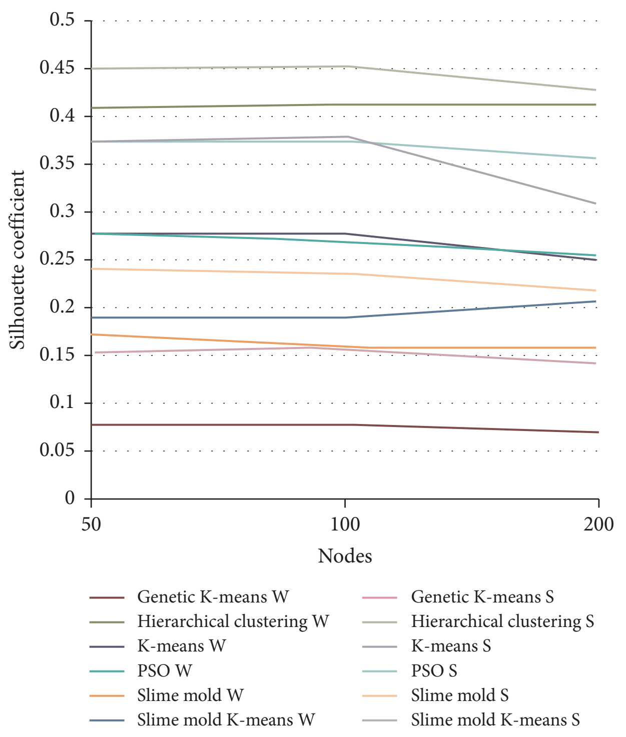

Figure 13: Comparison of silhouette coefficient results between worker and storage peers for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means for all network size levels.

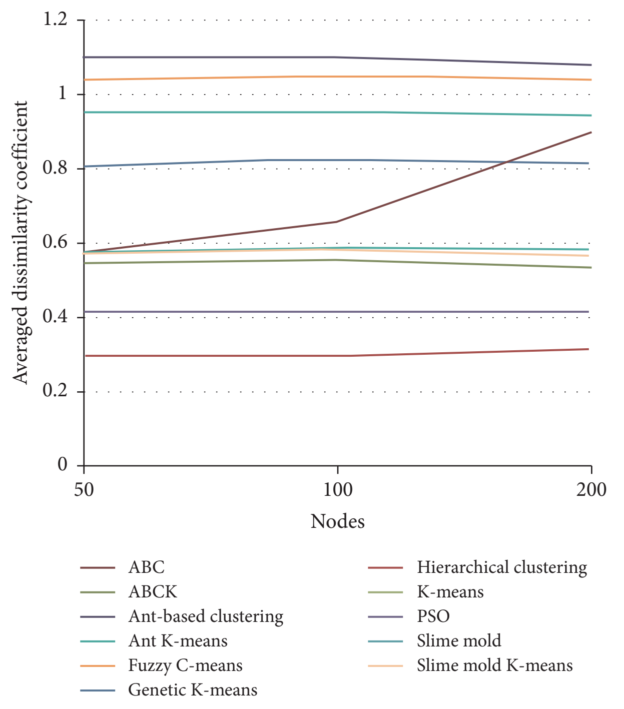

Figure 14: Averaged dissimilarity coefficient results for all network size levels.

has less worst case scenario solutions, i.e., low cohesion and well-separation. By contrast, genetic K-means has only an average decrease rate of 1%, whereas ant

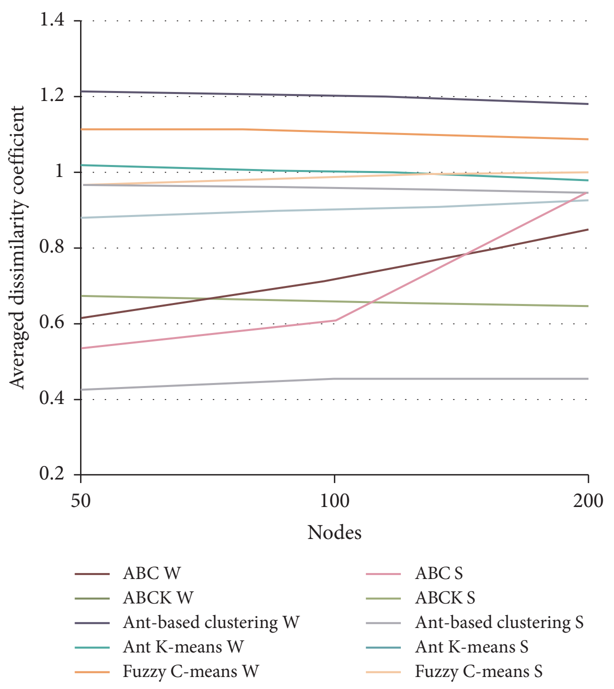

Figure 15: Comparison of averaged dissimilarity coefficient results between worker and storage peers for ABC, ABCK, ant-based clustering, ant K-means, and fuzzy C-means for all network size levels.

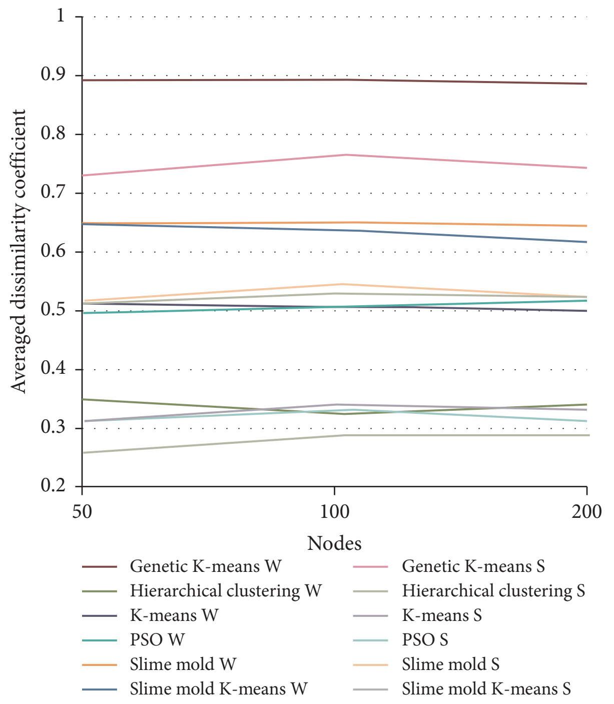

Figure 16: Comparison of averaged dissimilarity coefficient results between worker and storage peers for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means for all network size levels.

K-means algorithm first decreases by 6% (L_n to M_n), but then again increases by 4% (M_n to H_n). This indicates that genetic K-means algorithm is much more stable than ant K-means. Slime mold and slime mold K-means have very similar results for the DI. They have a decrease of around 7% from network size level L_n to M_n and a decrease of around 9% from network size level M_n to H_n. Fuzzy C-means and ABC share the last rank, showing a similar unstable behavior as ant K-means. Interestingly, K-means algorithm first exhibits a decrease of 9% (L_n to M_n), but then extraordinarily increases by 45% (M_n to H_n). Overall, it can be said that according to the DI results, every benchmarked algorithm performs by far better for storage peer clustering than for worker peer clustering.

Figure 9 presents the overall results for the SC for all benchmarked algorithms. In Figure 10, the comparison of the SC results between worker and storage peer clustering is depicted for ABC, ABCK, ant-based clustering, ant K-means, and fuzzy C-means, whereas in Figure 11, it is shown for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means.

The higher the SC is, the better the clustering result is. Hierarchical clustering increases by 0.5% (L_n to M_n) and then slightly decreases by 3% (M_n to H_n), whereas K-means algorithm has an increase of 0.4% (L_n to M_n) and then a decrease of 15% (M_n to H_n). Slime mold K-means algorithm outperforms slime mold by 12% (50 nodes), 18% (100 nodes), and 20% (200 nodes). Thus, slime mold K-means algorithm first slightly increases by about 2% from network size level L_n to M_n, and then decreases by 3% from network size level M_n to H_n, while slime mold has a decrease of 3% from network size level L_n to M_n and 5% from network size level M_n to H_n. This, also considering the results of the DBI and execution time, indicates that the application of K-means in the dispersion phase instead of the optimization by convergence to threshold value ε in the aggregation phase requires more time but seems to be more efficient. For the DBI results, the ant-based clustering algorithm also performs worst regarding the SC, which, as already mentioned, could be caused by its randomness. While fuzzy C-means algorithm starts off with the second-worst rank, it then massively increases by 42% (L_n to M_n) and 129% (M_n to H_n). The ABC algorithm starts with the third-worst result, but shows a drastical decrease of 121% (L_n to M_n) and 857% (M_n to H_n). Additionally, ABC is the only algorithm which shows better SC results for worker peer clustering than for storage peer clustering.

In Figure 12, the ADC results for all benchmarked algorithms are depicted. Figure 13 provides the comparison of the ADC results between worker peer clustering and storage peer clustering for ABC, ABCK, ant-based clustering, ant K-means, and fuzzy C-means, while Figure 14 provides it for genetic K-means, hierarchical clustering, K-means, PSO, slime mold, and slime mold K-means.

The smaller the ADC is, the better the clustering result is. Also, for the ADC, hierarchical clustering provides 25% (50 nodes), 28% (100 nodes), and 24% (200 nodes) and is more effective than its successor PSO, which is followed immediately by K-means with similar results. Slime mold and slime mold K-means provide similar results, while

Table 26: Slime mold ANOVA results.

	Mean ± stdev	p value	h
		50 nodes	
ABC	0.5755 ± 0.0008	0.790185171	0
ABCK	0.5470 ± 0.0008	0.073269084	0
Ant-based clustering	1.0986 ± 0.0007	5.77062E − 10	1
Ant K-means	0.9472 ± 0.0001	5.65653E − 10	1
Fuzzy C-means	1.0423 ± 0.0001	5.55717E − 11	1
Genetic K-means	0.8099 ± 0.0003	6.20493E − 08	1
Hierarchical clustering	0.2992 ± 0.0003	1.21269E − 08	−1
K-means	0.4095 ± 0.0005	1.84906E − 06	−1
PSO	0.3994 ± 0.0003	4.03503E-07	−1
Slime mold	0.5798 ± 0.0004	—	—
Slime mold K-means	0.5742 ± 0.0002	0.63865944	0
		100 nodes	
ABC	0.6583 ± 0.0003	0.000417622	1
ABCK	0.5540 ± 0.0001	0.001271288	−1
Ant-based clustering	1.1002 ± 0.00023	1.65439E − 11	1
Ant K-means	0.9531 ± 0.0002	2.38008E − 10	1
Fuzzy C-means	1.0476 ± 2.57904E-05	5.79645E − 12	1
Genetic K-means	0.8289 ± 0.0002	9.53206E − 09	1
Hierarchical clustering	0.3015 ± 0.0001	6.28362E − 10	−1
K-means	0.4210 ± 0.0001	3.0251E − 08	−1
PSO	0.4165 ± 0.0003	1.6151E − 07	−1
Slime mold	0.5959 ± 0.0002	—	—
Slime mold K-means	0.5810 ± 0.0001	0.130135878	0
		200 nodes	
ABC	0.8957 ± 0.0220	0.001496328	1
ABCK	0.5396 ± 0.0004	0.004216221	−1
Ant-based clustering	1.0802 ± 0.0003	3.19957E − 11	1
Ant K-means	0.9407 ± 2.90152E − 05	1.03927E − 11	1
Fuzzy C-means	1.0420 ± 1.15583E − 05	9.97955E − 13	1
Genetic K-means	0.8159 ± 0.0002	2.82205E − 09	1
Hierarchical clustering	0.3108 ± 7.25302E − 05	2.23741E − 10	−1
K-means	0.4109 ± 5.80434E − 05	6.94393E − 09	−1
PSO	0.4099 ± 7.33879E − 05	8.5432E − 09	−1
Slime mold	0.5809 ± 0.0002	—	—
Slime mold K-means	0.5685 ± 0.0001	0.136412571	0

both slightly have an increase of 3% and 1% for network size level L_n to M_n and then a decrease of 3% and 2% for network size level M_n to H_n, respectively. The ant-based clustering algorithm again performs worst regarding the ADC, just as for the DBI and the SC. While all benchmarked peer clustering algorithms scale rather fairly regarding the ADC, ABC falls out of line and increases massively by 14% (L_n to M_n) and 36% (M_n to H_n), which may be caused by the fact that the algorithm does not apply any local optimization such as an additional K-means step like its modification ABCK. Furthermore, ABC, just like all other algorithms, first shows better results for storage peer clustering than for worker peer clustering, but for 200 nodes, it has better performance when clustering worker peers.

4.7. Statistical Significance of the Results.

In order to evaluate and compare slime mold and slime mold K-means to the other benchmarked peer clustering algorithms profoundly, a statistical analysis as described in [7] is done. In detail, one-way ANOVA tests are performed with the following setup [7]: H_0 is the null hypothesis, which states that no significant difference between metric M of the two algorithms A and B exists, whereas H_1 is the corresponding alternative hypothesis. Consequently, if H_0 is rejected, there are not enough data available to determine whether algorithm A or B is significantly better than the other one. Furthermore, H_1 is concluded if H_0 is rejected [7].

Slime mold and slime mold K-means are tested separately for all combinations of network size levels (L_n, M_n, H_n) and exactly three queries and job requests. The ADC particularly focuses on the cohesion within the clusters. The significance level for the ANOVA tests is chosen as $\alpha = 0.05$. At first, slime mold is taking the role of algorithm A, whereas all other benchmarked algorithms, including slime mold K-means, embody algorithm B. Table 26 shows the results of these tests. After that, slime mold K-means algorithm embodies algorithm A, while all other benchmarked algorithms, including slime mold, take the role of algorithm B. For both result tables, it applies, if H_0 is concluded, the column h in the table has the value 0. If there is a significant

Table 27: Slime mold K-means ANOVA results.

	Mean ± stdev	p value	h
		50 nodes	
ABC	0.5755 ± 0.0008	0.927539959	0
ABCK	0.5470 ± 0.0008	0.095272996	0
Ant-based clustering	1.0986 ± 0.0007	2.14993E − 10	1
Ant K-means	0.9472 ± 0.0001	6.28834E − 11	1
Fuzzy C-means	1.0423 ± 6.00725E − 05	4.33966E − 12	1
Genetic K-means	0.8099 ± 0.0003	1.15244E − 08	1
Hierarchical clustering	0.2992 ± 0.0003	3.06358E − 09	−1
K-means	0.4095 ± 0.0005	8.33325E-07	−1
PSO	0.3994 ± 0.0003	1.16171E − 07	−1
Slime mold	0.5798 ± 0.0003	0.63865944	0
Slime mold K-means	0.5742 ± 0.0002	—	—
		100 nodes	
ABC	0.6583 ± 0.0003	4.63782E − 05	1
ABCK	0.5540 ± 0.0001	0.005729103	−1
Ant-based clustering	1.1002 ± 0.0002	4.18507E − 12	1
Ant K-means	0.9531 ± 0.0002	5.35059E − 11	1
Fuzzy C-means	1.0476 ± 2.57904E − 05	5.68231E − 13	1
Genetic K-means	0.8289 ± 0.0002	2.01672E − 09	1
Hierarchical clustering	0.3015 ± 0.0001	2.38163E − 10	−1
K-means	0.4210 ± 0.0001	1.38983E − 08	−1
PSO	0.4165 ± 0.0003	1.36687E − 07	−1
Slime mold	0.5959 ± 0.0002	0.130135878	0
Slime mold K-means	0.5810 ± 0.0001	—	—
		200 nodes	
ABC	0.8957 ± 0.0220	0.001170781	1
ABCK	0.5396 ± 0.0004	0.018074686	−1
Ant-based clustering	1.0802 ± 0.0003	1.49938E − 11	1
Ant K-means	0.9407 ± 2.90152E − 05	1.54553E − 12	1
Fuzzy C-means	1.0420 ± 1.15583E − 05	1.29002E − 13	1
Genetic K-means	0.8159 ± 0.0002	8.03541E − 10	1
Hierarchical clustering	0.3108 ± 7.25302E − 05	8.94089E − 11	−1
K-means	0.4109 ± 5.80434E − 05	3.17742E − 09	−1
PSO	0.4099 ± 7.33879E − 05	4.31318E − 09	−1
Slime mold	0.5809 ± 0.0002	0.136412571	0
Slime mold K-means	0.5685 ± 0.0001	—	—

Table A12: Abbreviations used in Tables A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, and A11.

Abbreviation	Explanation
Type	Specifies whether the results are from worker peer clustering, storage peer clustering, or the average of both
Nodes	Number of worker peer, storage peer, and client peer nodes participating in the benchmark
Time	Average execution time of one algorithm execution in seconds. Rounded to 2 decimals
DBI	Average Davies–Bouldin index of one clustering result. Rounded to 4 decimals
DI	Average Dunn index of one clustering result. Rounded to 4 decimals
SC	Average silhouette coefficient of one clustering result. Rounded to 4 decimals
ADC	Average averaged dissimilarity coefficient of one clustering result. Rounded to 4 decimals

difference between the tested peer clustering algorithms, i.e., H_0 is rejected, h takes the value of 1 if algorithm A performs significantly better than algorithm B; otherwise, h has the value of −1.

The test proves that there exists a significant difference of the ADC metric values. Therefore, slime mold performs significantly better than ABC at this configuration and h is set to 1. As shown in Table 25, while for a network size of

50 nodes, no significant difference between slime mold and ABC and ABCK, respectively, can be determined, for all other benchmarked algorithms, the result is the same for all network size levels. As can be seen in Table 27, slime mold K-means algorithm achieves rather similar results, but exactly the same final outcomes regarding significant differences. Also, on each network size level, no significant difference between slime mold and slime mold K-means exists.

5. Conclusion

Managing data in P2P systems is complex and challenging due to the network's large scale and the highly transient nature of its peers. In addition to clustering data, grouping peers based on their interests or data into peer clusters is beneficial. Peer clustering enhances querying performance and results in higher quality outcomes. Beyond clustering data, organizing peers into clusters based on their interests or data is advantageous. This peer clustering improves querying performance and yields higher quality results.

Certain characteristics of P2P systems make clustering particularly challenging. The high dynamics of P2P environments necessitate an intelligent, adaptable, and robust solution. Consequently, the problem of peer clustering is unique and must address all the issues inherent to the environment. This requires advanced, intelligent, and self-organizing approaches capable of handling the system's dynamics.

This paper proposes an innovative approach to effective peer clustering based on swarm intelligence, introducing two novel algorithms: slime mold and slime mold K-means, which mimic the life cycle of the Dd slime mold. These algorithms are benchmarked, evaluated, and compared against several well-known conventional and swarm-based algorithms, including ABC, ABC combined with K-means, ant-based clustering, ant K-means, fuzzy C-means, genetic K-means, hierarchical clustering, K-means, and PSO. All implemented algorithms are thoroughly benchmarked and evaluated.

5.1. Discussion and Evaluation. The slime mold and slime mold K-means algorithms exhibit strong performance in terms of execution time, both generally and specifically when compared to other swarm intelligence algorithms. These algorithms outperform all other swarm-inspired algorithms in both execution time and clustering solution quality. Although typically mid-range in their results, they avoid the unwanted significant increases or decreases in clustering effectiveness seen with other peer clustering algorithms. They scale well in terms of both execution time and effectiveness, maintaining consistent performance. In contrast, many other algorithms either display undesirable fluctuations in effectiveness or show great effectiveness with poor execution time, or vice versa.

Limitations of the current studies include benchmarking on middle scale tests and no churn. Benchmarks in larger environments would be of great interest. Additionally, the fluctuation of node participation (churn) is not simulated during the benchmark execution.

5.2. Future Work. Future work includes

- Combination of different algorithms, for example, [45–50]: an investigation of a hybrid approach.
- Combination of different approaches: a topic of interest is to investigate the effectiveness of combining peer clustering with load balancing and load clustering, as it would offer even more sophisticated ways of resource allocation.
- Regarding the implementation of slime mold and slime mold K-means, the following aspects could lead to an improvement when being researched:
    - o Vegetative movement: In the vegetative movement phase, amoebas are initialized randomly, i.e., the given number of cluster means is initialized randomly, as it is done for many other algorithms. An interesting approach to analyze would be to investigate the algorithms' behavior and performance when using a more sophisticated way of cluster mean initialization.
- Combination with hierarchical clustering: Although the slime mold combination with K-means shows fairly satisfactory results, a combination of slime mold with hierarchical clustering would be an interesting research issue.
- Investigating abovementioned limitations of the current studies.
- A detailed statistical analysis of the obtained results that takes in consideration (besides ANOVA) also some other tests, e.g., Friedman test [51, 52].

Appendix

A. Raw Result Data

In the following tables, the raw result data of the competitive benchmarks are shown. The tables contain the average result for each network size and show the separate results for worker peer clustering, storage peer clustering, and the average of both. For reasons of representation, the metrics used are abbreviated. Table A12 provides the corresponding explanations.

B. Distance Measurement Calculation

P2P distance: An example of how exactly the calculation is performed is provided below.

Worker Peer 1 (p): ("sorting, optimization", 4.4)
Worker Peer 2 (q): ("sorting", 1.2)

$$\text{MSD} = \sqrt{\left(\text{levenstein}\left(\text{sorting, optimization}'',''\text{sorting}''\right)\right)^2 + \left(\frac{4.4 - 1.2}{4.4 + 1.2/2}\right)^2} = \sqrt{(0.5)^2 - (1.14)^2} = 1.24. \tag{B.1}$$

Peer-to-cluster distance: The calculation of the distance between a peer and a cluster mean is illustrated below.

Cluster (p): ("sorting: 2, optimization: 1", 4.4)
Worker Peer (q): ("sorting", 1.2)

$$\text{MSD} = \sqrt{\left(\text{levenstein}\left(\text{sorting, sorting, optimization}'',''\text{sorting}''\right)\right)^2 + \left(\frac{4.4 - 1.2}{4.4 + 1.2/2}\right)^2} = \sqrt{(0.33)^2 - (1.14)^2} = 1.19. \tag{B.2}$$

Cluster-to-cluster distance: This calculation is illustrated by the example given below.

Cluster (p): ("sorting: 2, optimization: 1", 4.4)
Cluster (q): ("validity: 1", 1.2)

$$\text{MSD} = \sqrt{\left(\text{levenstein}\left(\text{sorting, sorting, optimization}'',''\text{validity}''\right)\right)^2 + \left(\frac{4.4 - 1.2}{4.4 + 1.2/2}\right)^2} = \sqrt{(1)^2 - (1.14)^2} = 1.52. \tag{B.3}$$

To demonstrate the calculation of the **P2P distance** using the Jaccard distance measurement, an example is given below:

WorkerPeer1 (p): ("sorting,optimization", 4.4)
WorkerPeer2 (q): ("sorting", 1.2)

$$\begin{aligned} J(p,q) &= \frac{d_J(p,q) + pd(p,q)}{2} = \frac{1 - |\{\text{sorting, optimization}\} \cap \{\text{sorting}\}|/2 + |4.4 - 1.2|/4.4 + 1.2/2}{2} \\ &= \frac{1 - |\{\text{sorting}\}|/|\{\text{sorting, optimization}\}| + 1.14}{2} = \frac{1 - 1/2 + 1.14}{2} = \frac{0.5 + 1.14}{2} = 0.82. \end{aligned} \tag{B.4}$$

The calculation of the **peer-to-cluster distance** using the Jaccard distance is illustrated by the example given below:

Cluster (p): ("sorting:2,optimization:1", 4.4)
WorkerPeer (q): ("sorting", 1.2)

$$\begin{aligned} J(p,q) &= \frac{d_J(p,q) + pd(p,q)}{2} = \frac{1 - |\{\text{sorting, optimization}\} \cap \{\text{sorting}\}|/|\{\text{sorting, optimization}\} \cup \{\text{sorting}\}| + |4.4 - 1.2|/4.4 + 1.2/2}{2} \\ &= \frac{1 - |\{\text{sorting}\}|/|\{\text{sorting, optimization}\}| + 1.14}{2} = \frac{1 - 1/2 + 1.14}{2} = \frac{0.5 + 1.14}{2} = 0.82. \end{aligned} \tag{B.5}$$

An example how the **cluster-to-cluster distance** is performed is provided below:

Cluster (p): ("sorting:2,optimization:1", 4.4)
Cluster (q): ("validity:1", 1.2)

$$\begin{aligned} J(p,q) &= \frac{1 - |\{\text{sorting, optimization}\} \cap \{\text{validity}\}|/|\{\text{sorting, optimization}\} \cup \{\text{validity}\}| + |4.4 - 1.2|/4.4 + 1.2/2}{2} \\ &= \frac{1 - (|\{\}|/|\{\text{sorting, optimization, validity}\}|) + 1.14}{2} = \frac{1 - 0/3 + 1.14}{2} = \frac{1 + 1.14}{2} = 1.07. \end{aligned} \tag{B.6}$$

C. A Sample: Ant-Based Clustering Versus ACO

Ant-based clustering and ACO are both inspired by the behavior of ants, but they are used for different purposes and work in different ways.

Ant-based clustering is a method inspired by the natural clustering behavior of ants, particularly how they group similar items (like food or brood) together. It is used for data clustering and pattern recognition. The idea is to group similar data points together based on certain features, mimicking how ants cluster similar objects in nature.

A mechanism can be described in the following way: In ant-based clustering, artificial ants move through a virtual environment where data items (representing objects, documents, etc.) are scattered. Ants pick up and drop items based on local similarity measures (e.g., similarity to nearby items). Over time, similar items are grouped together as ants repeatedly move items around, leading to the emergence of clusters.

ACO is a general-purpose optimization algorithm inspired by the foraging behavior of ants, particularly how they find the shortest path to food sources by laying down pheromones. ACO is used for solving different combinatorial optimization problems where the goal is to find the optimal solution from a large set of possibilities.

A mechanism can be described in the following way: Artificial ants construct solutions by moving through a solution space, guided by pheromone trails (which represent accumulated knowledge of good solutions) and heuristic information. Ants probabilistically choose paths based on the amount of pheromone present and the quality of the solution associated with those paths. Over time, paths that lead to better solutions get reinforced with more pheromone, while less optimal paths evaporate (lose pheromone concentration), helping ants to converge on the best solution.

Data Availability Statement

The data used to support the findings of this study are openly available in Repositum TUWien at https://repositum.tuwien.at/handle/20.500.12708/1844.

Conflicts of Interest

The authors declare no conflicts of interest.

Funding

No funding was received to assist with the preparation of this manuscript.

Acknowledgments

We would like to thank Laura Toifl for collaboration and constructive feedback [33].

Endnotes

[1]Note: There are many innovative, attractive algorithms that could be potentially used like [45–50]. However, they should be either clustering algorithms or should be carefully considered for further modeling and modification for clustering.

[2]According to Bloom's taxonomy.

[3]A general, reusable architectural solution that contributes to solving the communication and coordination generics found in a certain use case.

[4]https://www.sciencedirect.com/topics/mathematics/euclidean-distance.

[5]https://www.mathsisfun.com/percentage-difference.html.

[6]https://www.statisticshowto.com/jaccard-index/.

[7]https://repositum.tuwien.at/handle/20.500.12708/18441.

[8]Google Compute Engine. https://cloud.google.com/compute/.

References

[1] C. H. Ding, S. Nutanong, and R. Buyya, "Peer-to-Peer Networks for Content Sharing," *Peer-to-Peer Computing: The Evolution of a Disruptive Technology* (Hershey, PA: IGI Global, 2004), 28–65.

[2] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-To-Peer Content Distribution Technologies," *ACM Computing Surveys* 36, no. 4 (2004): 335–371.

[3] V. Šešum-Čavić and E. Kühn, "Self-Organized Load Balancing through Swarm Intelligence," *Next Generation Data Technologies for Collective Computational Intelligence* (Berlin, Heidelberg: Springer, 2011), 195–224.

[4] E. Kühn, A. Marek, T. Scheller, V. Šešum-Čavić, M. Vögler, and S. Craß, "A Space-Based Generic Pattern for Self-Initiative Load Clustering Agents," in *Coordination Models and Languages* (Heidelberg: Springer Berlin, 2012), 230–244.

[5] A. Modarresi, H. Mamat, and N. Mustapha, "A Social Network Peer-To-Peer Model for Peer Clustering," *2008 International Symposium on Information Technology* 3 (2008): 1–7.

[6] A. Singh and M. Haahr, "Decentralized Clustering in Pure P2P Overlay Networks Using Schelling's Model," *IEEE International Conference on Communications* (2007): 1860–1866.

[7] V. Šešum-Čavić, E. Kühn, and D. Kanev, "Bio-Inspired Search Algorithms for Unstructured P2P Overlay Networks," *Swarm and Evolutionary Computation* 29 (2016): 73–93.

[8] V. Šešum-Čavić, E. Kühn, and S. Zischka, "Swarm-Inspired Routing Algorithms for Unstructured P2P Networks," *International Journal of Swarm Intelligence Research, IJSIR* 9, no. 3 (2018): 2.

[9] V. Šešum-Čavić, "A Survey of Swarm-Inspired Metaheuristics in P2P Systems: Some Theoretical Considerations and Hybrid Forms," *International Journal of Swarm Intelligence* 5, no. 2 (2020): 244–282.

[10] K. Krishna and M. Murty, "Genetic K-Means Algorithm," *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics: A Publication of the IEEE Systems, Man, and Cybernetics Society* 29 (1999): 433–439.

[11] J. C. Bezdek, R. Ehrlich, and W. Full, "FCM—the Fuzzy C-Means Clustering- Algorithm," *Computers & Geosciences* 10 (1984): 191–203.

[12] J. C. Dunn, "A Fuzzy Relative of the Isodata Process and Its Use in Detecting Compact Well-Separated Clusters," *Journal of Cybernetics* 3, no. 3 (1973): 32–57.

[13] K. C. Gowda and G. Krishna, "Agglomerative Clustering Using the Concept of Mutual Nearest Neighbourhood," *Pattern Recognition* 10 (1978): 105–112.

[14] K. C. Gowda and T. Ravi, "Divisive Clustering of Symbolic Objects Using the Concepts of Both Similarity and Dissimilarity," *Pattern Recognition* 28 (1995): 1277–1282.

[15] K. Jain, M. N. Murty, and P. J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys* 31, no. 3 (1999): 264–323.

[16] P.-N. Tan, M. Steinbach, and V. Kumar, "Cluster Analysis: Basic Concepts and Algorithms," *Introduction to Data Mining*, 1st edition (Boston, MA: Addison-Wesley Longman Publishing Co., Inc, 2005).

[17] A. Adamatzky, "Route 20, Autobahn 7, and Slime Mold: Approximating the Longest Roads in USA and Germany With Slime Mold on 3-D Terrains," *IEEE Transactions on Cybernetics* 44, no. 1 (2014).

[18] E. Kühn and V. Šešum-Čavić, *A Framework-Based Approach for Flexible Evaluation of Swarm-Intelligent Algorithms, Women in Computational Intelligence, Key Advances and Perspectives on Emerging Topics* (Berlin, Germany: Springer, 2022).

[19] R. Kessin, *Dictyostelium: Evolution, Cell Biology, and the Development of Multicellularity* (Cambridge, UK: Cambridge Univ. Press, 2001).

[20] A. Crespo and H. Garcia-Molina, "Semantic Overlay Networks for P2P Systems," *Agents and Peer-to-Peer Computing*, 1–13 (Berlin, Heidelberg: Springer, 2005).

[21] M. Khambatti, K. D. Ryu, and P. Dasgupta, "Structuring Peer-To-Peer Networks Using Interest-Based Communities," *Databases, Information Systems, and Peer-To-Peer Computing* (Heidelberg: Springer Berlin, 2004), 48–63.

[22] B. Liang, J. Tang, J. Li, and K. Wang, "Keyword Extraction Based Peer Clustering," *Grid and Cooperative Computing-GCC 2004* (Heidelberg: Springer Berlin, 2004), 827–830.

[23] X. M. Huang, C. Y. Chang, and M. S. Chen, "Peer Cluster: A Cluster-Based Peer-To-Peer System," *IEEE Transactions on Parallel and Distributed Systems*, 17, no. 10, 1110–1123.

[24] N. Hashemipour, P. Crespo del Granado, and J. Aghaei, "Dynamic Allocation of Peer-To-Peer Clusters in Virtual Local Electricity Markets: A Marketplace for EV Flexibility," *Energy* 236 (2021).

[25] G. C. Okwuibe, A. S. Gazafroudi, E. Mengelkamp, S. Hambridge, P. Tzscheutschler, and T. Hamacher, "Advanced Clustering Approach for Peer-To-Peer Local Energy Markets Considering Prosumers' Preference Vectors," *IEEE Access* 11 (2023): 33607–33627.

[26] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics - Doklady* 10 (1966): 707–710.

[27] D. Karaboga and C. Ozturk, "A Novel Clustering Approach: Artificial Bee Colony (Abc) Algorithm," *Applied Soft Computing* 11, no. 1 (2011): 652–657.

[28] G. Armano and M. Farmani, "Clustering Analysis With Combination of Artificial Bee Colony Algorithm and K-Means Technique," *International Journal of Computer Theory and Engineering* 6 (2014): 141–145.

[29] E. D. Lumer and B. Faieta, "Diversity and Adaptation in Populations of Clustering Ants," in *3rd International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3: From Animals to Animats 3, SAB94* (Cambridge, MA: MIT Press, December 1994), 501–508.

[30] R. Kuo, H. Wang, T.-L. Hu, and S. Chou, "Application of Ant K-Means on Clustering Analysis," *Computers & Mathematics with Applications* 50, no. 10 (2005): 1709–1724.

[31] M. Omran, A. Salman, and A. Engelbrecht, "Image Classification Using Particle Swarm Optimization," *Simulated Evolution and Learning* 1 (2004): 370–374.

[32] R. H. Kessin, "Dictyostelium: Evolution, Cell Biology, and the Development of Multicellularity," *38 of Developmental and Cell Biology Series* (Cambridge, UK: Cambridge University Press, 2001).

[33] L. Fagagnini, "A Framework for Self-Initiative Peer Clustering Agents," (Vienna, Austria: Technische Universität Wien, 2021), Diploma Thesis, https://doi.org/10.34726/hss.2021.66923.

[34] D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence* no. 2 (1979): 224–227.

[35] J. C. Dunn, "Well-separated Clusters and Optimal Fuzzy Partitions," *Journal of Cybernetics* 4, no. 1 (1974): 95–104.

[36] P. J. R. Silhouettes, "A Graphical Aid to the Interpretation and Validation of Cluster Analysis," *Journal of Computational and Applied Mathematics* 20 (1987): 53–65.

[37] L. Kaufman and P. Rousseeuw, "Finding Groups in Data: An Introduction to Cluster Analysis," *Wiley Series in Probability and Statistics* (Hoboken, NJ: John Wiley & Sons, Inc, 1990).

[38] E. Kühn, S. Craß, G. Joskowicz, A. Marek, and T. Scheller, "Peer-Based Programming Model for Coordination Patterns," in *Coordination Models and Languages* (Heidelberg, Germany: Springer Berlin, 2013), 121–135.

[39] E. M. Kuehn, "A Practical Tool-Chain for the Development of Coordination Scenarios," in *Coordination Models and Languages. COORDINATION 2021. Lecture Notes in Computer Science*, eds. F. Damiani and O. Dardha, 12717 (Berlin, Germany: Springer, 2021).

[40] E. M. Kuehn, "The Peer-Model Tool-Chain," *Science of Computer Programming* 223 (2022): 102876.

[41] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari, "The Irace Package: Iterated Racing for Automatic Algorithm Configuration," *Oper. Res. Perspect.* 3 (2011): 43–58.

[42] U. Boryczka, "Ant Clustering Algorithm," *Intelligent Information Systems, 1998* (Singapore: Springer Verlag, 2008).

[43] J. Handl and B. Meyer, "Ant-Based and Swarm-Based Clustering," *Swarm Intelligence* 1, no. 2 (2007): 95–113.

[44] J. C. Bezdek, *Pattern Recognition With Fuzzy Objective Function Algorithms* (Norwell, MA: Kluwer Academic Publishers, 1981).

[45] J. Chen, Y. Guo, M. S. M. Kasihmuddin, et al., "MTS-PRO2SAT: Hybrid Mutation Tabu Search Algorithm in Optimizing Probabilistic 2 Satisfiability in Discrete Hopfield Neural Network," *Mathematics* 12 (2024): 721.

[46] Y. Guo, N. E. Zamri, M. S. M. Kasihmuddin, et al., "Dual Optimization Approach in Discrete Hopfield Neural Network," *Applied Soft Computing* 164 (2024).

[47] C. Zheng, M. S. M. Kasihmuddin, M. A. Mansor, J. Chen, and Y. Guo, "Intelligent Multi-Strategy Hybrid Fuzzy K-Nearest Neighbor Using Improved Hybrid Sine Cosine Algorithm," *Mathematics* 10 (2022): 3368.

[48] S. Kumar, D. Datta, and S. K. Singh, "Black Hole Algorithm and Its Applications," in *Computational Intelligence Applications in Modeling and Control. Studies in Computational Intelligence*, eds. A. Azar and S. Vaidyanathan, 575 (Berlin, Germany: Springer, 2015).

[49] S. Mirjalili and A. Lewis, "The Whale Optimization Algorithm," *Advances in Engineering Software* 95 (2016): 51–67.

[50] M. Sidik, N. E. Zamri, M. S. M. Kasihmuddin, H. A. Wahab, Y. Guo, and M. A. Mansor, "Non-Systematic Weighted Satisfiability in Discrete Hopfield Neural Network Using Binary Artificial Bee Colony Optimization," *Mathematics* 10 (2022): 1129.

[51] N. E. Zamri, M. A. Mansor, M. S. M. Kasihmuddin, et al., "A Modified Reverse-Based Analysis Logic Mining Model With Weighted Random 2 Satisfiability Logic in Discrete Hopfield Neural Network and Multi-Objective Training of Modified Niched Genetic Algorithm," *Expert Systems with Applications* 240 (2024).

[52] N. E. Zamri, S. A. Azhar, M. A. Mansor, A. Alway, and M. S. M. Kasihmuddin, "Weighted Random K Satisfiability for K=1,2 (r2SAT) in Discrete Hopfield Neural Network," *Applied Soft Computing* 126 (2022).