applied sciences

MDPI

Article

Application of Multi-Objective Hyper-Heuristics to Solve the Multi-Objective Software Module Clustering Problem

Haya Alshareef and Mashael Maashi *

Department of Software Engineering, College of Computer and Information Sciences, King Saud University, Riyadh 11451, Saudi Arabia; 441204617@student.ksu.edu.sa
* Correspondence: mmaashi@ksu.edu.sa; Tel.: +966-11-805-58736

Abstract: Software maintenance is an important step in the software lifecycle. Software module clustering is a HHMO_CF_GDA optimization problem involving several targets that require minimization of module coupling and maximization of software cohesion. Moreover, multi-objective software module clustering involves assembling a specific group of modules according to specific cluster criteria. Software module clustering classifies software modules into different clusters to enhance the software maintenance process. A structure with low coupling and high cohesion is considered an excellent software module structure. In this study, we apply a multi-objective hyper-heuristic method to solve the multi-objective module clustering problem with three objectives: (i) minimize coupling, (ii) maximize cohesion, and (iii) ensure high modularization quality. We conducted several experiments to obtain optimal and near-optimal solutions for the multi-objective module clustering optimization problem. The experimental results demonstrated that the HHMO_CF_GDA method outperformed the individual multi-objective evolutionary algorithms in solving the multi-objective software module clustering optimization problem. The resulting software, in which HHMO_CF_GDA was applied, was more optimized and achieved lower coupling with higher cohesion and better modularization quality. Moreover, the structure of the software was more robust and easier to maintain because of its software modularity.

Keywords: multi-objective optimization problem; multi-objective evolutionary algorithms; software module clustering; multi-objective hyper-heuristics

Citation: Alshareef, H.; Maashi, M. Application of Multi-Objective Hyper-Heuristics to Solve the Multi-Objective Software Module Clustering Problem. *Appl. Sci.* **2022**, *12*, 5649. https://doi.org/10.3390/app12115649

Academic Editors: Sokratis Katsikas, José Salvador Sánchez Garreta and Stylianos Pappas

Received: 15 April 2022
Accepted: 31 May 2022
Published: 2 June 2022

1. Introduction

Software module clustering (SMC) classifies software modules into different clusters to enhance program structure and facilitate easy maintenance [1]. An excellent module software structure is one with low coupling and high cohesion [2]. Clustering helps discover groups and identify interesting distributions and patterns in the underlying data. Moreover, using some techniques (i.e., data mining) that analyze and extract novel, interesting patterns from data, clustering helps data mining techniques extract relations between software projects and extract information to assess the behavior of software projects. Some suitable clustering algorithms introduce optimal clustering techniques and suggest that it is essential to distinguish the consequences of heterogeneous clustering techniques. Several heterogeneous clustering techniques help solve the problem from different approaches, i.e., partition-based clustering, density-based clustering, and hierarchical clustering [3,4]. In this article, we attempted to solve the software module cluster by utilizing multi-objective optimization using hyper-heuristic methods to solve a wide range of problems in different domains.

The multi-objective optimization problem includes two or more objectives that demand to be minimized or maximized based on several related techniques presented for multi-objective optimization. Multi-objective evolutionary algorithm (MOEA) methods are commonly used to solve multi-objective optimization problems. The multi-objective SMC

problem can be described as assembling a specific group of modules into clusters according to given criteria [5]. It is also a search problem that is divided into two components. The first is creating a representation of the problem to explore methodologies, and the second is creating a cost function, or a fitness function, to estimate the quality of the solution [5]. Hyper-heuristics for multi-objective optimization problems is a new area of research in evolutionary computation and operational research [6,7]. Hyper-heuristic methods have been proposed to increase the generality of search methodologies [8]. Moreover, hyper-heuristics provide general search algorithms suitable for solving a wide range of problems in different domains [5,8–10]. Different heuristic components can be combined, selected, or generated in the hyper-heuristic method to efficiently solve a given optimization problem. On the one hand, hyper-heuristics methodologies involve a high-level strategy that controls the search over a set of heuristics and not controlling a search over a representation of the solutions directly [8]. On the other hand, hyper-heuristics with low-level heuristics using deterministic or non-deterministic methods perform as a "heuristic scheduler" [9]. Considering that each heuristic has its strengths and weaknesses, hyper-heuristics aim to automatically inform the algorithm by combining each heuristic's strengths and making up for the weaknesses of others. The hyper-heuristic multi-objective genetic algorithm can be applied to solve a multi-objective SMC problem as a multi-objective search problem. This study aims to solve the multi-objective module clustering optimization problem by using a multi-objective hyper-heuristic (MOHH_CF_GDA) approach [11] to minimize coupling, maximize cohesion, and deliver an optimal or near-optimal solution for the problem.

We addressed three main issues: (1) How can we achieve minimum coupling and maximum cohesion when solving the module clustering optimization problem? (2) How can we apply search-based software engineering techniques such as MOHH to solve our problem and deliver an optimal solution? (3) How can we interpret the solution and present it to the decision makers (i.e., the software engineer and the project manager)?

This study's major contributions are summarized as follows.

- Applying the MOHH_CF_GDA method to solve the multi-objective SMC optimization problem, which is considered as novel.
- Designing a framework for optimizing SMC to help the decision maker minimize the effort and time needed to maintain the software.
- Developing a special solution representation matrix to fit with the SMC optimization problem, and to facilitate applying the MOHH_CF_GDA method to achieve and deliver an optimal solution.
- Building a visual graph to represent the software module cluster before and after applying the MOHH_CF_GDA to display to the decision maker the optimal software module structure.

The rest of this article is structured as follows. Section 2 provides the background and related works. Section 3 presents the materials and the proposed methodology. Section 4 describes the experimental settings. Section 5 presents the results and discussions. Section 6 provides the conclusion and suggestions for further research.

2. Preliminaries and Related Work

2.1. Overview of SMC

SMC is a process of classifying software modules into different clusters to enhance program structure and facilitate easy maintenance [1]. An excellent module software structure is considered to be one that has low coupling and high cohesion [2]. An SMC framework is depicted in Figure 1. The software clustering process is conducted in several steps. The first step is extracting the module-level dependencies from the source code and storing the result in a database using an analysis tool for the source code. After the extracted model-level dependencies are stored in a database, the script is executed to query in the database. The query results are filtered, and a textual representation of the module is produced as a graphical representation comprising nodes and weighted directed

graphs. The former graphical representation displays the system modules, and the second representation shows the relations between modules [2].

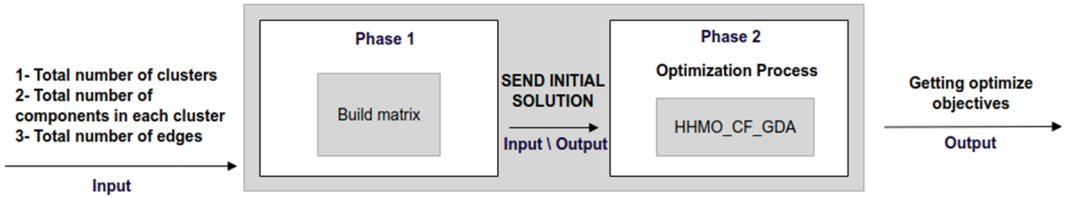


Figure 1. The framework of module clustering optimization prototype.

2.2. Search-Based Software Engineering

The notion of search-based software engineering (SBSE) was introduced by Herman and Jones in 2001 [12]. The term "search" indicates the meta-heuristic search-based optimization techniques used [12,13]. SBSE is relevant to software engineering (SE) and uses search-based optimization techniques to find and address various software engineering difficulties as optimization problems. SBSE optimization techniques are generally applied to solve optimization problems in several SE areas, such as software requirement engineering and software design, testing, refactoring, and management. Search-based optimization techniques seek to recognize and determine optimal solutions to a specific problem in a search space containing the candidate solutions. Generally, to solve a specific problem, the problem's representation should clearly define an objective function to evaluate the solutions based on their quality to achieve optimal solutions during the search process [13,14]. Search-based techniques are mostly applied to resolve many software engineering optimization issues. The most commonly used algorithms are genetic algorithms (GAs) [15], simulated annealing [16], particle swarm optimization [17], ant colony optimization [18] and hill climbing [19]. These algorithms are collectively known as evolutionary algorithms [12].

2.3. A Multi-Objective Hyper-Heuristic Approach

Hyper-heuristics have lately increased in use by researchers, and various papers on hyper-heuristics are still mainly limited to single-objective optimization. The use of hyper-heuristics for multi-objective optimization problems is a new area of research in evolutionary computation and operational research [6,7]. The multi-objective hyper-heuristic evolutionary algorithm (MHypEA) [20] has some features that provide excellent solutions for a wide range of optimization problems such as a generic form of low-level heuristics without combining any domain-specific knowledge, selection of low-level heuristics based on support learning linked with roulette-wheel selection, dynamic adaptation of the weights assigned to low-level heuristics based on the performance of the heuristics in directing the search towards the right areas and acceptable exploitation and exploration of the search space. Studies have been identified that deal with hyper-heuristics for multi-objective problems.

A paper entitled [1] Fast Multi-objective Hyper-heuristic Genetic Algorithm for Multi-Objective Software Module Cluster is presented. Six test problems were solved by using the two-archive multi-objective evolutionary algorithm. The second paper [21] presented a solution to the multi-objective software module clustering problem based on a hyper-heuristic approach using the multi-objective hyper-heuristic evolutionary algorithm (MHypEA). The problem was studied using two multi-objective approaches to clustering—the equal-size cluster approach (ECA) and the maximizing cluster approach (MCA). The MHypEA can be used to recommend suitable modular structures for weighted MDGs. In the third paper [1], a multi-objective hyper-heuristic genetic algorithm (MHypGA) is applied to solve the multi-objective software module clustering problem using selection, crossover, and mutation operations of evolutionary algorithms.

In the scientific literature, many studies use SBSE approaches to address software model clustering problems. In this article, we present the main ideas of SBSE techniques, including meta-heuristics and hyper-heuristics designed to solve multi-objective optimization problems. There are only a few studies on using MOHH to solve module clustering problems, which highlights the need for scientific research in the area of solving module clusters using multi-objective evolutionary algorithms and combining meta-heuristics to yield improved performance, rather than focusing on solving only one problem.

3. Materials and Methods

This study applied the multi-objective choice function-great deluge hyper-heuristic (HHMO_CF_GDA) approach proposed by Maashi et al. [5]. HHMO_CF_GDA controls and combines three multi-objective evolutionary algorithms including the multi-objective genetic algorithm (MOGA) [22,23], the non-dominated sorting genetic algorithm (NSGAII) [24,25], and the strength Pareto evolutionary algorithm (SPEA2) [26]—which acting as low-level heuristics in the hyper-heuristic framework [27]. As a selection method, we utilized a choice function with a high-level strategy that adaptively ranks the performance of three low-level heuristics, choosing which one to call at each decision point. Based on four performance metrics—algorithm effort [10], ratio of non-dominated individuals (RNI) [10], uniform distribution (UD) of a non-dominated population [24], and size of space covered (SSC; also known as S-metric hypervolume) [28]—we used an online learning mechanism to give knowledge of the problem domain to the choice mechanism [5]. The great deluge algorithm (GDA) acts as an acceptance criterion to accept or reject the candidate solution at each iteration

3.1. Problem Description and Formulation

The purpose of software modularity is to shape a number of subsystems by clustering the software elements in the software system. The consistency of the software module's clustering outcome can be evaluated in terms of cohesion and coupling. Cohesion is a function that tests how the modules in a cluster are interrelated. On the other hand, coupling is an inter-cluster term that tests how two given clusters are inter-edge dependencies. Coupling is as minimal as possible between the different subsystems, and cohesion within the subsystem is as high as possible [29]. Our goal is that a high-quality partition has both a low coupling relationship and a high cohesion relationship, which assumes that well-designed systems are formed by cohesive sets of loosely related modules between each other.

The multi-objective software module clustering problem can be described as a specific group of modules assembled into clusters according to given criteria [30]. The modules' relationships automatically enable better-quality clustering of software modules in the software module clustering problem. These relationships take the form of dependencies among modules. The idea is to minimize coupling between clusters and to maximize cohesion within each cluster. Clustering partitions the collection of all modules in the system. The collection of modules in each partition of the clustering is a cluster. Finding the most suitable clustering for a given collection of modules is an NP-hard problem, making it ideal for search-based software engineering techniques [30]. The clustering problem is defined as follows:

> *"The set of n objects X = {Xl, X2, ... , xn} is to be grouped. Each object, Xi, is to be clustered into non-overlapping groups 0 = {01, O2, ... , Ok} where OJ is a cluster j , or a set of objects and k is the number of clusters, 01 U O2 U . . . U Ok = X, 0i cJ ¢ and 0i n OJ = ¢ for i cJ j"* [30]

Clustering presents an idea of the properties of the groups rather than the individuals within them. Recently, clustering methods have been used to support comprehension of the software. The software module clustering problem can be established as a search problem and divided into two components—the first one is a representation of the problem for

exploring methodologies and the second one is a cost function, or a fitness function, to estimate the quality of solutions [30].

The software module optimization problem is to assign cohesion and coupling values, trying to achieve maximum cohesion and low coupling. We define the cohesion relationship A_i of cluster i with N_i components and μ_i intra-edge dependencies [31] as follows:

$$\text{Maximizing } A_i = \frac{\mu_i}{N_i^2}, \tag{1}$$

Coupling between the i-th cluster and the j-th cluster is expressed by $E_{i,j}$ after the software module is clustered [29]:

$$\text{Minimizing } E_{i,j} = \begin{cases} 0\,, & i = j \\ \frac{\varepsilon_{i,j}}{2N_iN_j}, & i \neq j \end{cases}, \tag{2}$$

We represent a system's modularization quality (MQ) as a function that shows the trade-off between interconnectivity and intra-connectivity [31]. Given a model dependency graph partitioned into k clusters, where A_i is the intra-connectivity of the i-th cluster, and E_{ij} is the interconnectivity between the i-th and j-th clusters, we represent MQ as follows:

$$MQ = \begin{cases} \frac{\sum_{i=1}^{k} A_i}{k} - \frac{\sum_{i,j=1}^{k} E_{ij}}{\frac{k(k-1)}{2}}, & \forall k > 1 \\ A_i, & k = 1 \end{cases}, \tag{3}$$

MQ establishes a trade-off between interconnectivity and intra-connectivity that remunerates the creation of highly cohesive clusters and penalizes excessive inter-edge dependencies. The average interconnectivity achieves this trade-off from the average intra-connectivity. MQ is bound between -1 (no cohesion inside the clusters) and 1 (no coupling among the clusters) [31]; see Table 1 for notation explanation.

Table 1. Notations for problem formulation.

Notations	Descriptions
N	Total number of components (nodes) in cluster i
μ_i	Number of intra-edge dependencies in cluster i
ε	Number of inter-edge dependencies in clusters i and j
N_iN_j	Total number of components (nodes) in clusters i and j

In this study, we applied HHMO_CF_GDA [5] to solve our multi-objective module clustering problem. The HHMO_CF_GDA framework proposed in Maashi [5,11,32] is shown in Figure 2. The high-level strategy can be a learning mechanism or a meta-heuristic [33]. The high-level strategy task is to lead the search effectively and adapt based on the success or failure of the low-level heuristic components during the search process to reuse the applied method for solving several problems [34]. Low-level heuristics are the hyper-heuristic framework's problem domain-specific elements; therefore, they can enter any related information, such as candidate solutions. So, the high-level strategy does not alter while low-level heuristics and the evaluation function need editing when taking a new problem [33]. In the HHMO_CF_GDA framework, there is a clear separation between the high-level hyper-heuristic method and low-level heuristic components. The idea of the domain barrier is to provide a higher level of abstraction for hyper-heuristics and raise hyper-heuristics' generality by applying it to a new problem without changing the framework. The barrier domain only supports problem information, such as fitness, cost, and penalty value [35]. This barrier does not allow any problem-specific information to pass to the high-level strategy during the search process. The framework is designed

in the same modular manner, making it highly flexible and easily replaceable, and its components reusable. The multi-objective choice function-great deluge hyper-heuristic (HHMO_CF_GDA) controls and combines the robustness of three multi-objective evolutionary algorithms (NSGAII, SPEA2, and MOGA), which can be used as low-level heuristics. As a selection method, the choice function utilized with a high-level strategy adaptively ranks the performance of three low-level heuristics, choosing which one to call at each decision point. The great deluge algorithm (GDA) is employed as a move acceptance based on four performance metrics—AE [10], RNI [10], SSC [28], and UD [24]. An online learning mechanism is used to obtain knowledge of the problem domain for the selection mechanism [5].

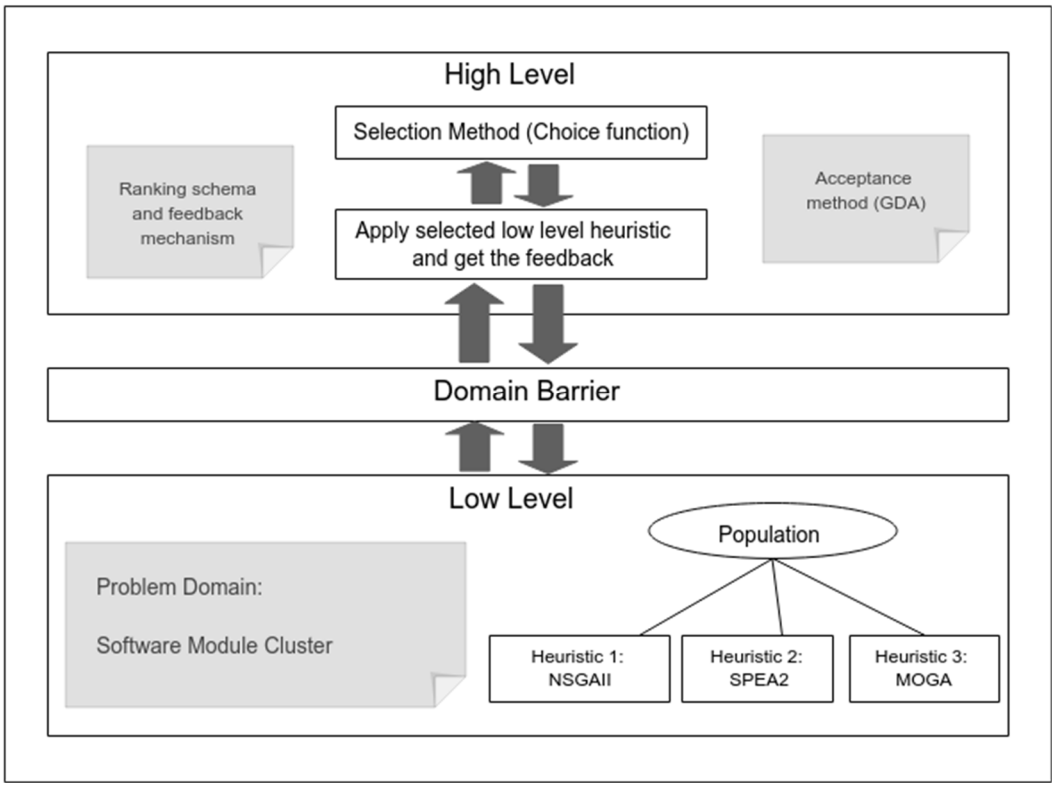


Figure 2. The multi-objective choice function-great deluge hyper-heuristic (HHMO_CF_GDA) framework. Readapted from [5].

We propose a module clustering optimization framework as shown in Figure 1. HHMO_CF_GDA was used to solve the software module clusters in order to find optimal or near-optimal solutions that meet the three objectives (low coupling, high cohesion, and high modularization quality). As shown in Figure 1, the module clustering optimization framework consists of two phases. Phase 1 involves building the matrix. In this phase, the software module matrix is constructed based on the total number of clusters, the number of components in each cluster, and the number of edges between them. The built matrix output is an initial solution individual with appropriate representations based on the total numbers of components, clusters, and edges. The output of phase 1 acts as an input for the optimization process in phase 2. In this phase, we apply the HH_CF_GDA optimizer to solve the multi-objective module clustering optimization problem. The output-optimized solution is built as a diagram to increase decision maker understandability.

The pseudocode of the multi-objective module clustering optimization framework based on HHMO_CF_GDA is shown in Algorithm 1. The SMC framework consists of two phases; the first phase aims to build the input matrix using software information from the decision maker including input values, the total number of clusters, the total number of components in each cluster, and the total number of edges. Then, the number

of edges between each node is counted until the total number of nodes is achieved. After determining each cluster with its edges and components, we calculate cohesion, coupling, and initial MQ values. Then, those values are fed to the optimization phase and act as inputs to the HHMO_CF_GDA method. In the second phase, HHMO_CF_GDA is run to solve the SMC problem and obtain the optimized solution.

Algorithm 1 Module Cluster Optimization	
Input	Total number of clusters, total number of components in each cluster, and total number of edges
Result	Getting optimize objectives
1.	**Phase 1: Build matrix**
1.1	Sorting each cluster with its edges and components
1.2	Calculte the objective's value and send it as an initial value
2.	**Phase 2: Optimization process**
2.1	Apply the optimization process (HHMO_CF_GDA)
2.2	Getting optimize objectives.

HHMO_CF_GDA acts as the optimizer in phase2. The pseudocode of HHMO_CF_GDA [5] is reprinted in Algorithm 2. HHMO_CF_GDA combines three multi-objective meta-heuristics as low-level heuristics for solving a multi-objective optimization problem, which in our case is a module clustering optimization problem. HHMO_CF_GDA performs a fixed number of iterations. Initially, all low-level heuristics are run for a fixed number of function evaluations with the same population size and number of generations. Then, all low-level heuristics are ranked with respect to performance metrics (AE, RNI, SCC, and UD). The low-level heuristic with the best performance is selected to execute the next iteration. In each iteration, one low-level heuristic is run and then the ranking of all low-level heuristics is updated. This process is reputed until the stopping criteria are met. The choice function provides a balance between intensification and diversification. The choice function addresses the trade-off between the undiscovered areas of the search space and the past performance of each heuristic. So, the heuristic with the best performance will be chosen more frequently to exploit the search area. This will boost the intensification element. The time element in the choice function boosts diversification. A low-level heuristic that is executed for a long period of time is recalled to support diversification by exploring unvisited areas of the search space. See [5] for more details on how HHMO_CF_GDA works.

Algorithm 2 HHMO_CF_GDA	
Input Data	HHMO_CF_GDA (H, F) whereas H = low level heuristic, F = performance metrics
Result	Getting optimized result
1.	**Initialization (take initial objectives values)**
1.1	Run all H members
1.2	For all H members, get the values of all members of F for each member of H
1.3	Rank all H member based on ranking scheme
1.4	Get the values of the simple choice function for each member of H.
1.5	The member of H with the largest choice function value will be selected as an initial heuristic
2.	**Repeat**
4.	Execute the selected member of H.
5.	Get the values of all member of F for the selected H member
6.	Update the rank for all H members based on ranking scheme
7.	Updating the choice function values for all H members.
8.	The member of H with the largest choice function value will be selected for the next iteration
10.	**Until stopping condition is met**

3.2. Solution Representation

We encoded the chromosomes in a way that represents the main characteristic of our multi-objective SMC that can be applied to the multi-objective choice function-based hyper-heuristic method. We used the binary bit string, which is primarily used to encode the chromosomes. In the module clustering optimization problem, each chromosome represents one solution for the clustering modules, as shown in Table 2. A gene in a chromosome represents one cluster, and each gene comprises the following elements: total number of nodes (TON) and total number of edges for each node (TOE). Moreover, each individual chromosome represents a solution of the module clustering problem; each solution consists of a set of genes, and each gene represents one component, which consists of a total number of edges. Before running HHMO_CF_GDA, the built matrix constructs the initial solution individuals with appropriate representations based on the total numbers of components, clusters, and edges.

Table 2. Representation of the clustering module solution.

1 chromosome (1 individual)									
C1(TON)	C1(TOE)	C2(TON)	C2(TOE)	C3(TON)	C3(TOE)	C4(TON)	C4(TOE)	. . . ..	CN(TON)	CN(TOE)
gene1		gene2		gene3		gene4			geneN	

Table 3 represents sample data for the input matrix used to save the number of edges between each node until the total number of nodes is achieved. Table 4 represents each node, which belongs to a specific cluster (label) which indicates the total number of clusters.

Table 3. The built input matrix.

Node	Main	User	Control	State	Family	Computer
Main	0	1	1	0	0	0
User	1	0	0	1	1	1
Control	1	0	0	0	1	0
State	0	1	0	0	1	1
Family	0	1	1	1	0	0
Computer	0	1	0	1	0	0
System	0	1	1	1	0	1

Table 4. Clustering labels.

Main	User	Control	State	Family	Computer	System
Cluster1	Cluster2	Cluster1	Cluster3	Cluster2	Cluster4	Cluster3

4. Experiments and Settings

A set of experiments were conducted using two datasets to observe the differences between the performances of each multi-objective meta-heuristic (NSGAII, SPEA2, and MOGA) and HHMO_CF_GDA [5] in solving the multi-objective module clustering problem. The two datasets were mutins [20], which is an operating system for educational purposes written in the Turing language and comprises 20 components and 57 edges, and telnet2 [36], which comprises 28 components and 81 edges. Table 5 presents the datasets. We implemented module clusters in C++ using the Microsoft Visual Studio C++ platform and Ubuntu and performed 30 independent runs on the datasets using Linux x86_64 and Windows 10 with Intel Core i7 computer systems. The HHMO_CF_GDA algorithm executed 100 iterations with a population size of 32 and 50 generations in each run. We

chose 100 iterations (i) to show the heuristic with the best performance for many iterations due to the impact of the intensification factor, and (ii) to provides the opportunity for other heuristics to be called because of the diversification factor. For fair comparison, each low-level heuristic was used in isolation and terminated after 1600 × 100 evaluation functions. For each dataset, 30 independent trials were run for each algorithm with different random seeds and a population size of 32 and 50 generations in each run. For all three low-level heuristics, the crossover and mutation probabilities were set to 0.9 and 1/24, respectively, and the corresponding distribution indices were set to 10 and 20, respectively. Distance sharing σ for the UD metric and MOGA was set to 0.01 in the normalized space. This setting was used for SSC and UD as a feedback indicator in the ranking scheme of HHMO_CF_GDA and as a performance measure for comparison. All the algorithms were implemented with the same common sub-functions using Microsoft Visual Studio C++ platform and Ubuntu on Windows 10 with Intel Core i7 computer system. Furthermore, all the experimental parameters were the same as those commonly used in the scientific literature for discrete problems [37]. Table 6 presents the parameter settings used to solve the module cluster problem.

Table 5. Dataset information.

Dataset	Total Number of Components	Total Number of Edges
mutins	20	57
telnet2	28	81

Table 6. Parameter settings of HHMO_CF_GDA and low-level heuristics.

HHMO_CF Parameters	Settings
Population size	32
Total number of generations	50
Total number of iterations of HHMO_CF_GDA	100
Stopping criteria	Max iterations
Length of individual chromosome	Based on total number of components
Number of independent runs	30
Random seed for NSGAII2	1.0
Number of independent runs for NSGAII2	Based on ranking scheme
Number of independent runs for SPEA2	Based on ranking scheme
Random seed for SPEA2	11
Number of independent runs for MOGA	Based on ranking scheme

5. Results and Discussion

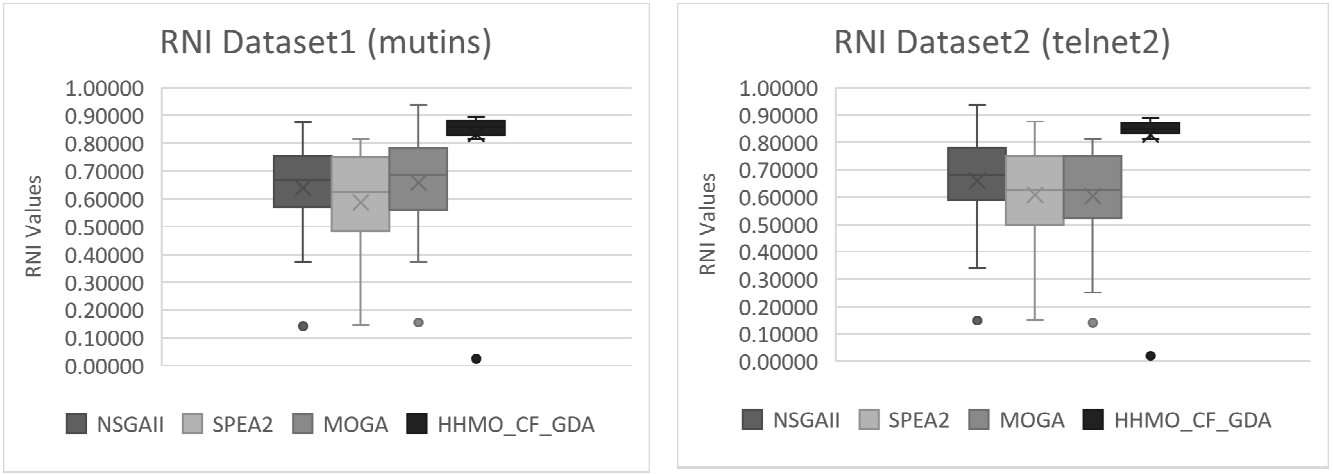
We conducted various experiments to achieve an optimal or near-optimal solution for the multi-objective module cluster problem. We repeated each experiment 30 times for each dataset and calculated the average, maximum, minimum, and standard deviation of the results. For all the performance metrics, a higher value indicates better performance.

For each dataset, HHMO_CF_GDA exhibited a better performance compared to the individual low-level algorithms with respect to RNI, SSC, and UD, as presented in Table 7. The results revealed that HHMO_CF_GDA yielded a better solution than the other algorithms. The performance results are also displayed as box plots in Figures 3–5 to provide a clear visualization of the distributions of the simulation data of the 30 independent runs. The results revealed that HHMO_CF_GDA achieved the maximum possible cohesion and minimum coupling, considering MQ, to keep the result as balanced as possible compared to the standalone algorithms. A higher SSC value indicates a better quality of

non-dominated space, that is, a smaller distance from the actual Pareto optimal front. The SSC metric calculates the size of the objective function space occupied by the solutions surrounding the Pareto optimal front. The higher the UD value, the better the quality of the non-dominated set, that is, the non-dominated front is spread widely along the Pareto optimal front. The UD metric evaluates the distribution of non-dominated individuals over the Pareto optimal front. RNI values lie between 0 and 1, and the metric evaluates the fraction of non-dominated individuals in the population and indicates the quality of the solution. A RNI value of 0 indicates that none of the individuals in a population are non-dominated, whereas a RNI value of 1 indicates that all the individuals in a population are non-dominated. Figures 6 and 7 show the successful solution obtained when the HHMO_CF_GDA approach was applied to the mutins dataset. Figure 6 shows the structure of the mutins software before the application of HHMO_CF_GDA, whereas Figure 7 shows the optimized mutins software result of the module clustering after the application of HHMO_CF_GDA. It was observed that the structure of the mutins software was more optimized after applying HHMO_CF_GDA. In addition, the nodes and clustering were more organized, cohesion of the software clusters was higher, and coupling between the nodes was lower. Similarly, HHMO_CF_GDA successfully achieved the objectives after it was applied to optimize the telnet2 software. Figure 8 shows the telnet2 dataset before the application of HHMO_CF_GDA, whereas Figure 9 shows the optimized telnet2 dataset results of the module clustering after the application of HHMO_CF_GDA.

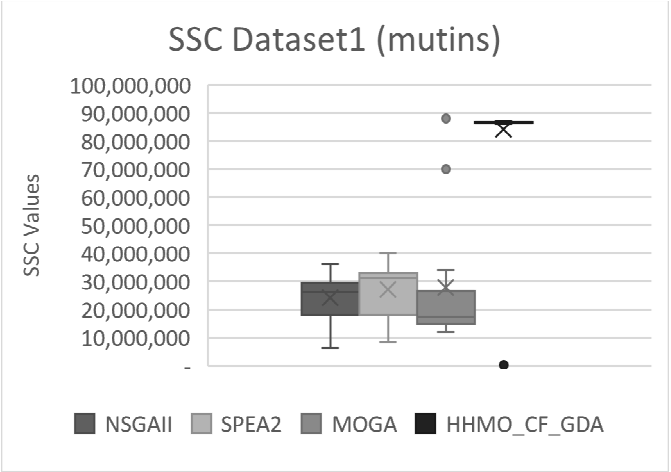
Table 7. Average performance of HHMO_CF_GDA compared to low-level heuristics on the mutins and telnet2 datasets with respect to the ratio of non-dominated individuals (RNI), size of space covered (SSC), and uniform distribution (UD).

Dataset	Method	RNI				SSC				UD			
		AVG	MIN	MAX	STD	AVG	MIN	MAX	STD	AVG	MIN	MAX	STD
Dataset 1 (mutins)	NSGAII	0.65671	0.37500	0.87500	0.142272338	24795565	13,000,000	36,100,000	6,258,335	0.41061	0.17536	0.64427	0.1307
	SPEA2	0.60532	0.25000	0.81250	0.147015408	27846667	13,000,000	40,000,000	8,331,442	0.45983	0.17536	0.68357	0.1460
	MOGA	0.67414	0.37500	0.93750	0.154935576	26531667	12,000,000	90,000,000	23,804,432	0.33415	0.16575	0.61257	0.1234
	HHMO_CF_GDA	**0.85727**	0.81606	0.89466	**0.025291224**	**86591950**	86,180,700	86,963,900	**268,330**	**0.61415**	0.58441	0.63763	0.0155
Dataset 2 (telnet2)	NSGAII	0.67662	0.34028	0.93750	0.148507718	32460000	12,000,000	90,000,000	25,474,959	0.33816	0.16480	0.61257	0.11512
	SPEA2	0.62795	0.18750	0.87500	0.151444811	27973333	13,000,000	90,000,000	18,705,945	0.37900	0.16575	0.67008	0.1412
	MOGA	0.62396	0.25000	0.81250	0.13938832	32746667	11,000,000	90,000,000	20,866,451	0.42805	0.17536	0.68357	0.1427
	HHMO_CF_GDA	**0.85264**	0.81400	0.88908	**0.01992233**	**86591757**	86,225,400	86,955,900	**195,271**	**0.61202**	0.58296	0.64152	0.01947

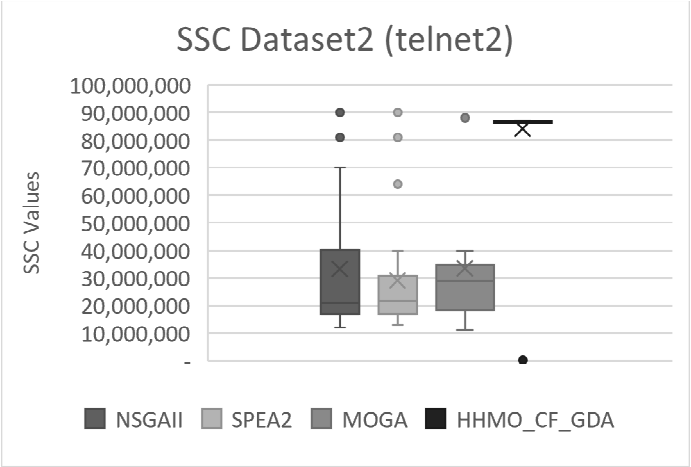


(**a**) mutins (**b**) telnet2

Figure 3. Box plots of NSGAII, SPEA2, MOGA, and HHMO_CF_GDA of the ratio of non-dominated individuals (RNI) on the mutins and telnet2 datasets: (**a**) mutins box plots; (**b**) telnet2 box plots.

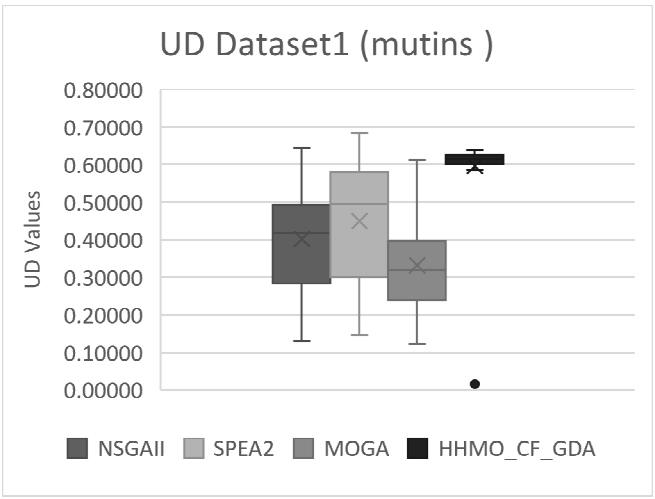


(**a**) mutins

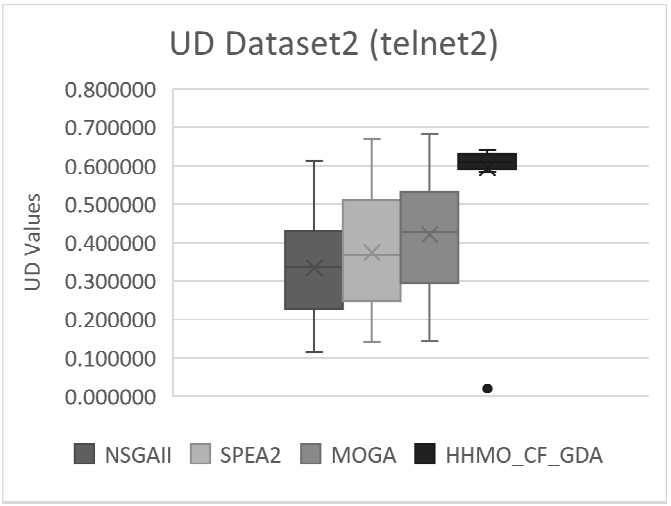


(**b**) telnet2

Figure 4. Box plots of NSGAII, SPEA2, MOGA, and HHMO_CF_GDA of the size of space covered (SSC) on the mutins and telnet2 datasets: (**a**) mutins box plots; (**b**) telnet2 box plots.



(**a**) mutins



(**b**) telnet2

Figure 5. Box plots of NSGAII, SPEA2, MOGA, and HHMO_CF_GDA of uniform distribution (UD) on the mutins and telnet2 datasets: (**a**) mutins box plots; (**b**) telnet2 box plots.

From the results, we made two observations: (a) HHMO_CF_GDA showed its applicability in a wide range of problems in different domains. In this study, HHMO_CF_GDA was applied to solve the multi-objective software module clustering problem, which is a real-world problem in discrete space. The algorithm effectively achieved the objectives of higher cohesion and lower coupling. Additionally, an increase in the generality of the HHMO_CF_GDA methodology, achieving diversity and convergence, was attainted. This observation supports HHMO_CF_GDA results when applied to solve the vehicle crashworthiness design problem, which is a real-world problem in continuous space [32]. (b) HHMO_CF_GDA can efficiently solve different optimization problems regardless of the kind of problem space, whether in discrete space, as in the case of multi-objective software module clustering, or in continuous space, as in the case of the vehicle crashworthiness design problem.

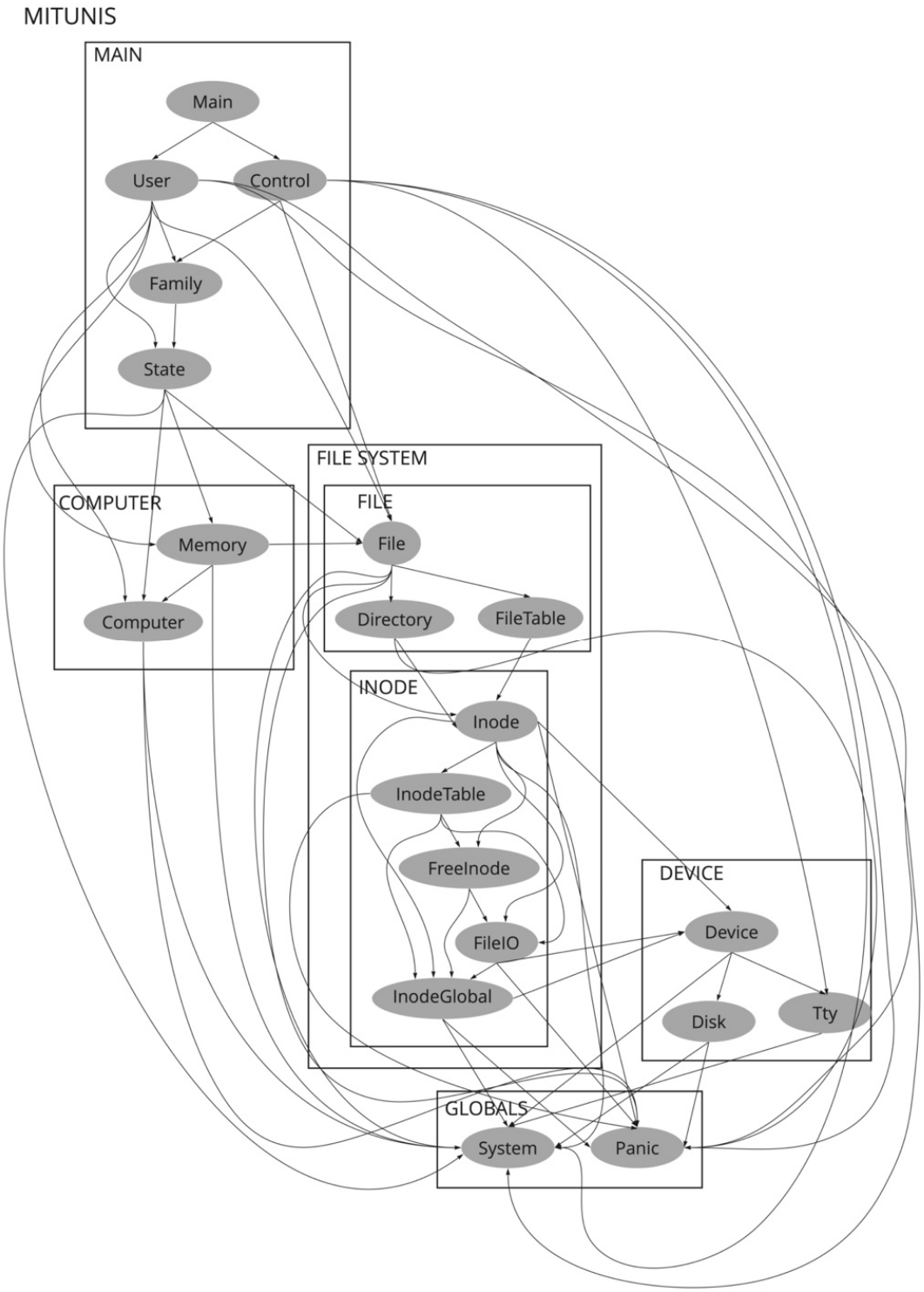


Figure 6. Mutins system before applying the HHMO_CF_GDA approach. Adapted from [31].

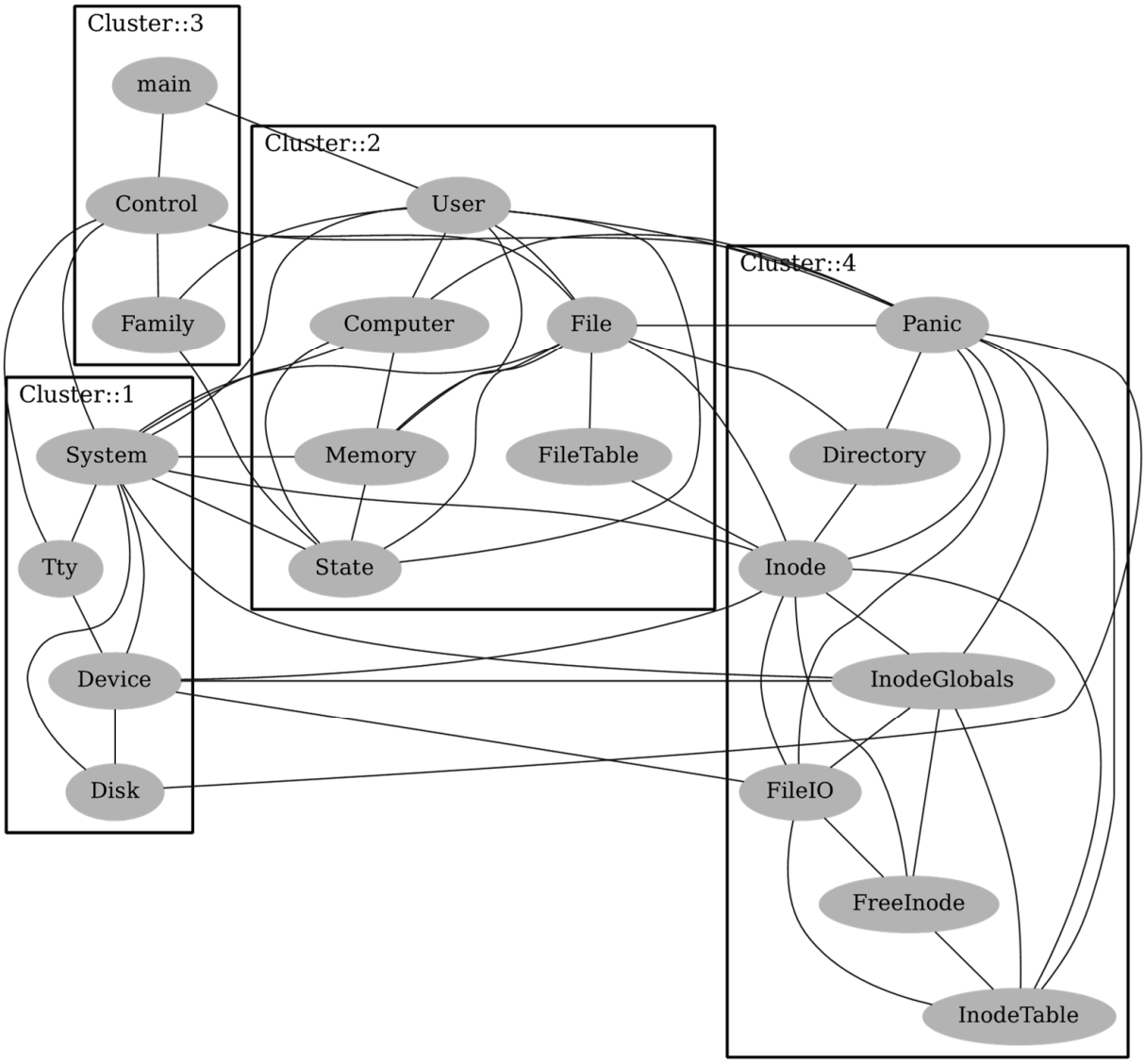


Figure 7. Mutins system after applying the HHMO_CF_GDA approach.

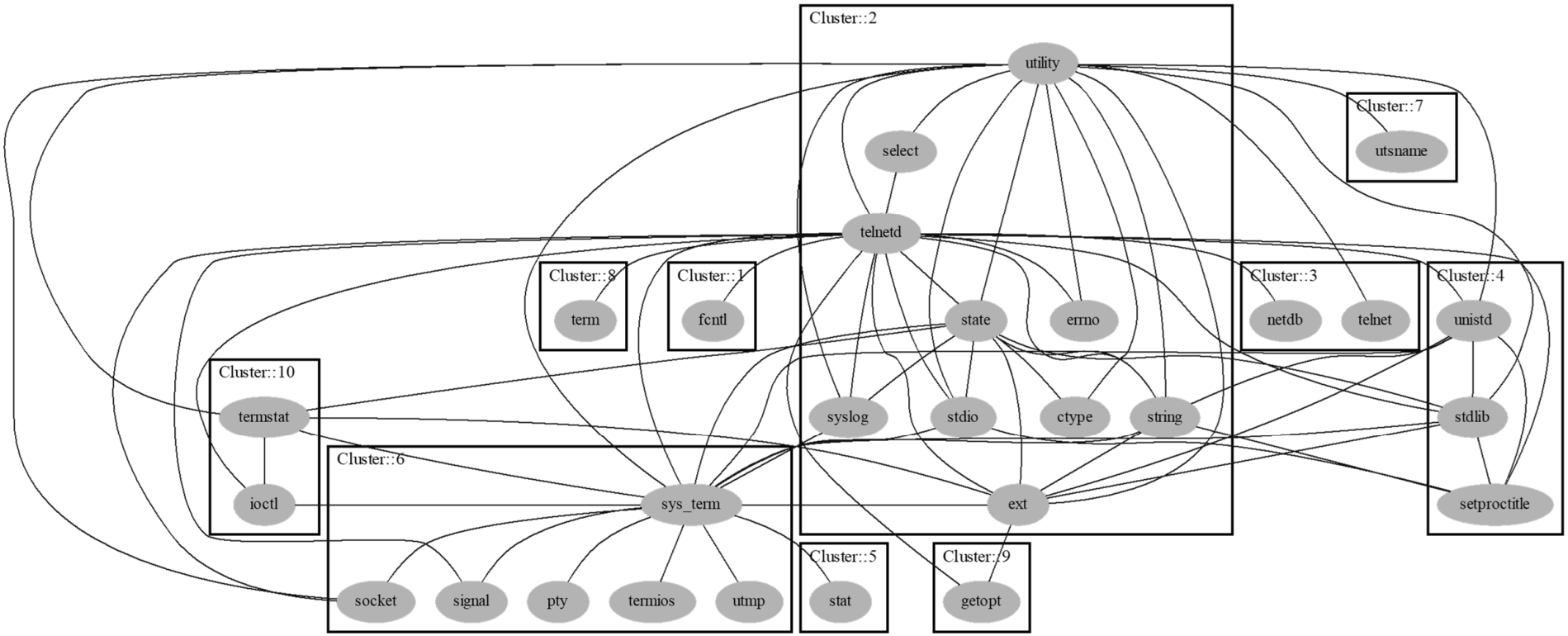


Figure 8. Telnet2 system before applying the HHMO_CF_GDA approach.

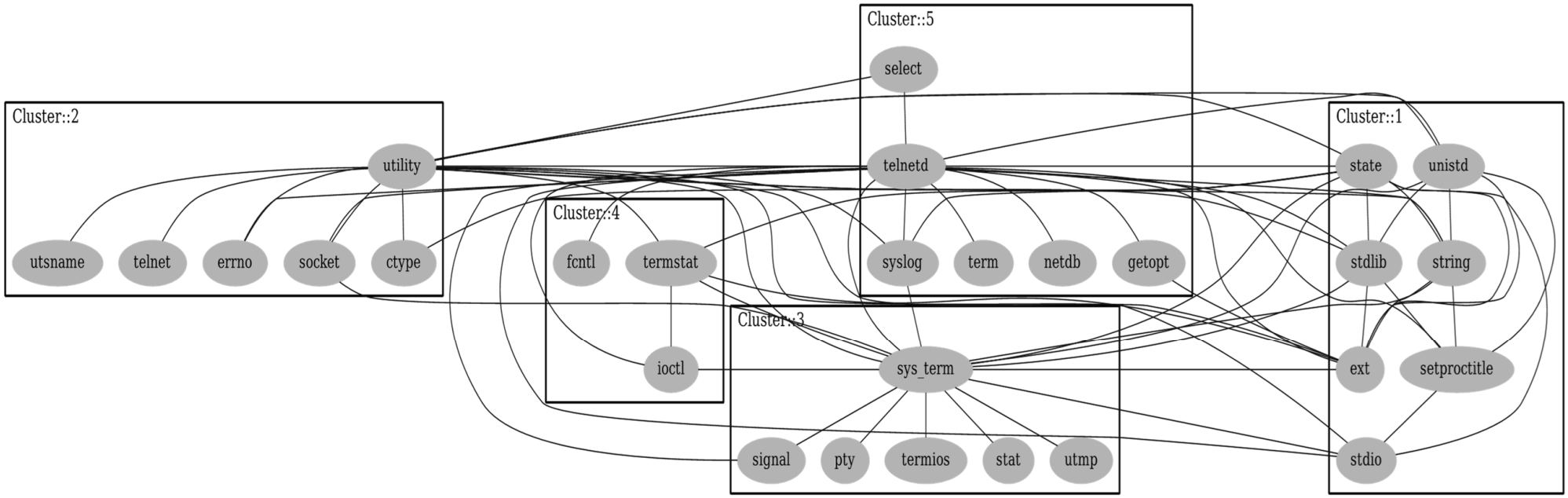


Figure 9. Telnet2 system after applying the HHMO_CF_GDA approach.

6. Conclusions

The need for systems that can help software engineers in decision making to optimize SMC for their systems motivated us to develop a HHMO_CF_GDA approach to solve the multi-objective module clustering optimization problem. This approach can help software engineers to optimize module clustering to minimize coupling and maximize cohesion. We also introduced a multi-objective module clustering framework that comprises two phases: building the input matrix and running the optimization process. In the matrix-building phase, the software structure is obtained from the decision maker and then proceeds to the next phase to run the optimization framework (HHMO_CF_GDA). The optimizer minimizes coupling and maximizes cohesion by considering MQ objective, which is to balance coupling and cohesion. Subsequently, the proposed framework produces the optimized solution as a diagram to make it more understandable to the decision maker. We apply HHMO_CF_GDA to solve the multi-objective module clustering optimization problem. We also conducted several independent experiments on mutins and telnet2 datasets using HHMO_CF_GDA and the individual low-level heuristics of MOEA algorithms to determine the performance of HHMO_CF_GDA. The experimental results show that the HHMO_CF_GDA method outperformed the individual multi-objective evolutionary algorithms in solving the multi-objective software module clustering optimization problem. The resulting software, in which HHMO_CF_GDA was applied, was more optimized and achieved lower coupling with higher cohesion and better modularization quality. Moreover, the structure of the software was more robust and easier to maintain because of its software modularity. As a future work, we intend to add a multi-objective evolutionary algorithm based on the decomposition algorithm instead of MOGA to HHMO_CF_GDA to improve its performance in solving the module clustering optimization problem. More optimized software with high modularity can lead to more maintainable software. To enhance the modularity of the software in the optimization process, some possible methods can be applied, such as the k-means-like clustering algorithm based on the silhouette analysis approach [38], to estimate the optimal number of clusters in the clustering categorical data. The algorithm uses the kernel-density estimation approach to define the cluster centers for the clustering step. It also uses information theory-based dissimilarity to measure the distance between the centers and objects in each cluster. Moreover, it evaluates the quality of the cluster obtained in the first step to select the best k. Another algorithm is the hierarchical agglomerative clustering algorithm for restructuring software systems (HARS) [39], which identifies the refactoring required to restructure a software system. The HARS algorithm is used to obtain an improved software system structure by identifying the required refactoring.

Author Contributions: Data curation, H.A.; formal analysis, M.M. and H.A.; investigation, M.M. and H.A.; methodology, H.A.; project administration, M.M.; resources, M.M.; software, H.A.; supervision, M.M.; writing—original draft, H.A.; writing—review and editing, M.M. All authors will be informed about each step of manuscript processing including submission, revision, and revision reminders via emails from our system or the assigned Assistant Editor. All authors have read and agreed to the published version of the manuscript.

Funding: The authors received funding from the Deanship of Scientific Research at King Saud University for this study. H. Alshareef. https://dsrs.ksu.edu.sa/, accessed on 1 May 2022.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are available on request from the corresponding author, [M. Maashi].

Acknowledgments: The authors thank the Deanship of Scientific Research at King Saud University for funding and supporting this research through the initiative of DSR Graduate Students' Research Support.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

HHMO_CF	Choice Function-Based Hyper-Heuristic for Multi-Objective
MOEA	Multi-Objective Evolutionary Optimization
MOEA\D	Multi-Objective Algorithm Based on Decomposition
MQ	Modularization Quality
NSGAII	Non-Dominated Sorting Genetic Algorithm
SBSE	Search-Based Software Engineering
SE	Software Engineering
SMC	Software Module Clustering
SPEA2	Strength Pareto Evolutionary
MOHH	Multi-Objective Hyper-Heuristic
GA	Genetic Algorithms
HHMO_CF_GDA	Multi-Objective Choice Function-Great Deluge Hyper-Heuristic
RNI	Ratio of Non-Dominated Individuals
SSC	Size of Space Covered
UD	Uniform Distribution
GDA	Great Deluge Algorithm
TON	Total Number of Nodes
TOE	Total Number of Edges for Each Node

References

1. Kumari, A.; Srinivas, K.; Gupta, M. Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. In Proceedings of the 2013 3rd IEEE International Advance Computing Conference (IACC), Ghaziabad, India, 22–23 February 2013; pp. 813–818.
2. Kumari, A.C.; Srinivas, K. Software module clustering using single and multi-objective approaches. *Int. J. Adv. Res. Comput. Sci. Softw. Eng. Technol.* **2013**, *5*, 12–18.
3. Seetharaman, S.; Thouheed Ahmed, S.; Gunashree, P.B.; Ishwarya, B.A. A Generalized Study on Data Mining and Clustering Algorithm. In Proceedings of the International Conference On Computational Vision and Bio Inspired Computing, Coimbatore, India, 29–30 November 2018; pp. 1121–1129.
4. Xie, T.; Thummalapenta, S.; Lo, D.; Liu, C. Data Mining for Software Engineering. *Computer* **2009**, *42*, 55–62. [CrossRef]
5. Maashi, M. An Investigation of Multi-Objective Hyper-Heuristics for Multi-Objective Optimization. Ph.D. Thesis, University of Nottingham, Nottingham, UK, 2014.
6. Bai, R.; Van Woensel, T.; Kendall, G.; Burke, E.K. A new model and a hyper-heuristic approach for two-dimensional shelf space allocation. *4OR* **2013**, *11*, 31–55. [CrossRef]
7. Özcan, E.; Bilgin, B.; Korkmaz, E. A comprehensive analysis of hyper-heuristics. *Intell. Data Anal.* **2008**, *12*, 3–23. [CrossRef]
8. Burke, E.K.; Hyde, M.; Kendall, G.; Ochoa, G.; Ozcan, E.; Woodard, J.R. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*; Springer: Boston, MA, USA, 2010; Volume 146, pp. 449–468.
9. Burke, E.; Kendall, G. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*; Springer: New York, NY, USA, 2005; ISBN 0-387-23460-8. [CrossRef]
10. Tan, K.; Lee, T.; Khor, E. Evolutionary algorithms for multi-objective optimization: Performance assessments and comparisons. In Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546), Seoul, Korea, 27–30 May 2001; Volume 2, pp. 979–986.
11. Maashi, M.; Kendall, G.; Özcan, E. A multi-objective hyper-heuristic based on choice function, Expert Systems with Applications. *Expert Syst. Appl.* **2014**, *41*, 4475–4493. [CrossRef]
12. Bibi, N.; Anwar, Z.; Ahsan, A. Comparison of Search-Based Software Engineering Algorithms for Resource Allocation Optimization. *J. Intell. Syst.* **2016**, *25*, 629–642. [CrossRef]
13. Harman, M.; Mansouri, S.; Zhang, Y. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.* **2012**, *45*, 1–61. [CrossRef]
14. Rezende, A.; Silva, L.; Britto, A.; Amaral, R. Software project scheduling problem in the context of search-based software engineering: A systematic review. *J. Syst. Softw.* **2019**, *155*, 43–56. [CrossRef]
15. Vega-Velázquez, M.; García-Nájera, A.; Cervantes, H. A survey on the Software Project Scheduling Problem. *Int. J. Prod. Econ.* **2018**, *202*, 145–161. [CrossRef]
16. Amine, K. Multiobjective Simulated Annealing: Principles and Algorithm Variants. *Adv. Oper. Res.* **2019**, *2019*, 8134674. [CrossRef]
17. Shi, Y.; Eberhart, R. Empirical study of particle swarm optimization. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 6–9 July 1999; Volume 3, pp. 1945–1950.

18. Dorigo, M.; Di Caro, G. Ant colony optimization: A new meta-heuristic. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 6–9 July 1999; Volume 2, pp. 1470–1477.
19. Chalup, S.; Maire, F. A study on hill climbing algorithms for neural network training. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 6–9 July 1999; pp. 2014–2021.
20. Holt, R. The UNIX System and Tunis. Reading. In *Concurrent Euclid*; Addison-Wesley: Boston, MA, USA, 1983.
21. Kumari, A.; Srinivas, K. Hyper-Heuristic Approach for Multi-Objective Software Module Clustering. *J. Syst. Softw.* **2016**, *117*, 384–401. [CrossRef]
22. Fonseca, C.; Fleming, P. Genetic Algorithms for Multiobjective Optimization: Formulation Discussion and Generalization. *ICGA J.* **1993**, *93*, 416–423.
23. Fonseca, C.; Fleming, P. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. *IEEE Trans. Syst. Man Cybern.—Part A Syst.* **1998**, *28*, 26–37. [CrossRef]
24. Srinivas, N.; Deb, K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.* **1994**, *2*, 221–248. [CrossRef]
25. Deb, K.; Goel, T. Controlled elitist non-dominated sorting genetic algorithms for better convergence. In Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization, Zurich, Switzerland, 7–9 March 2001; Springer: Berlin/Heidelberg, Germany; Volume 1993, pp. 67–81.
26. Zitzler, E.; Laumanns, M.; Thiele, L. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*; TIK. Report; ETH Library: Zürich, Switzerland, 2001; Volume 103, pp. 95–100.
27. Van Veldhuizen, D.; Lamont, G. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art, Evolutionary Computation. *Evol. Comput.* **2000**, *8*, 125–147. [CrossRef]
28. Zitzler, E.; Thiele, L. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Trans. Evol. Comput.* **1999**, *3*, 257–271. [CrossRef]
29. Sun, J.; Ling, B. Software module clustering algorithm using probability selection. *Wuhan Univ. J. Nat. Sci.* **2018**, *23*, 93–102. [CrossRef]
30. Praditwong, K. Solving software module clustering problem by evolutionary algorithms. In Proceedings of the Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE), Nakhonpathom, Thailand, 11–13 May 2011; pp. 154–159.
31. Doval, D.; Mancoridis, S.; Mitchell, B. Automatic clustering of software systems using a genetic algorithm. In Proceedings of the STEP '99. Proceedings Ninth International Workshop Software Technology and Engineering Practice, Pittsburgh, PA, USA, 2 September 1999; pp. 73–81. [CrossRef]
32. Maashi, M.; Özcan, E.; Kendall, G. Choice function based hyper-heuristics for multi-objective optimization. *Appl. Soft Comput.* **2015**, *41*, 312–326. [CrossRef]
33. Burke, E.; Kendall, G.; Newall, J.; Hart, E.; Ross, P.; Schulenburg, S. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of Metaheuristics*; Springer: Boston, MA, USA, 2003; pp. 457–474.
34. Qu, R.; Burke, E. Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. *J. Oper. Res. Soc.* **2009**, *60*, 1273–1285. [CrossRef]
35. Hussin, N. Tabu Search Based Hyper-Heuristic Approaches for Examination Timetabling School of Computer Science and Information Technology. Ph.D. Thesis, University of Nottingham, Nottingham, UK, 2005.
36. npm. Available online: https://www.npmjs.com/package/telnet2 (accessed on 16 April 2021).
37. Huband, S.; Hingston, P.; Barone, L.; While, L. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Trans. Evol. Comput.* **2006**, *10*, 477–506. [CrossRef]
38. Dinh, D.; Fujinami, T.; Huynh, V. Estimating the Optimal Number of Clusters in Categorical Data Clustering by Silhouette Coefficient. In Proceedings of the International Symposium on Knowledge and Systems Sciences, Da Nang, Vietnam, 29 November–1 December 2019; Springer: Singapore, 2019; Volume 1103, pp. 1–17, ISBN 978-981-15-1209-4.
39. Czibula, I. Hierarchical Clustering for Software Systems Restructuring. In Proceedings of the 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 21–24 September 2017; pp. 239–246. [CrossRef]