International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 5– No.6,April 2012 – www.ijais.org

Software Module Clustering using a Fast Multi-objective Hyper-heuristic Evolutionary Algorithm

A. Charan Kumari
Department of Physics & Computer Science
Dayalbagh Educational Institute
Dayalbagh, Agra, India

K. Srinivas
Department of Electrical Engineering
Dayalbagh Educational Institute
Dayalbagh, Agra, India

ABSTRACT

Software evolution is a natural phenomenon in the software development life cycle. As the software evolves, the modular structure of software degrades, and at one point it becomes a challenging task to maintain the software further. Software module clustering is an important activity during software maintenance whose main goal is to obtain good modular structures. Software engineers greatly emphasize on good modular structures as it is easier to comprehend, develop and maintain such software systems. In recent times, the problem has been converted into a Search-based Software Engineering Problem with multiple objectives. This problem is NP hard as it is an instance of graph partitioning and hence cannot be solved using traditional optimization techniques. The Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA) is a fast and effective metaheuristic search technique for suggesting software module clusters while maximizing cohesion and minimizing coupling of the software modules. It incorporates twelve low-level heuristics which are based on different methods of selection, crossover and mutation operations of Evolutionary Algorithms. The selection mechanism to select a low-level heuristic is based on reinforcement learning with adaptive weights. The effectiveness of the algorithm has been studied on six real-world module clustering problems reported in the literature and the comparison of the results prove the efficacy of the MHypEA in terms of quality of solutions and computational time.

General Terms

Software Engineering, Software maintenance.

Keywords

Search-based Software Engineering, Software module clustering, Hyper-heuristics, Evolutionary Algorithm.

1. INTRODUCTION

Software maintenance is a critical activity in the software development life cycle. Boehm [1] found through his research that the maintenance costs can be up to ten times those of an initial development and also Parikh and Zvegintzov [2] mentioned that it consumes 50% of all computer and human resources. Due to the increase in the complexity and size of the software systems, maintenance has become a crucial issue for software engineers. The situation becomes appalling, when the software lacks proper documentation on the changes that are performed during system evolution. As a consequence, it becomes practically difficult to maintain the software in future. The replacement of such software with a new one is also not a feasible option, as it does not guarantee the full functionality. The practices such as reverse engineering and reengineering have emerged to handle such software systems. As the source code is the only exact duplication of the system available to software developers and maintainers, the reverse engineering community is working towards the development of methods to extract the high-level structural information from the source code directly. Such methods are inclined to focus on design recovery through software clustering, program slicing, source code analysis etc.

Software module clustering is the process of grouping software modules into clusters in such a way that the highly dependent modules are grouped into the same cluster. The clustering assists in better comprehension of the system and provides easy maintenance in the future. This decomposition is based on the relationships among the modules. Generally these relationships are represented in the form of Module Dependency Graphs (MDGs), in which modules are represented as nodes and their inter relationships are represented by means of edges connecting the nodes. Thus clustering can be visualized as a graph partitioning problem, which is known to be NP hard. Therefore it cannot be solved efficiently by traditional optimization techniques. This led to the development of search-based approaches to the solution of software module clustering. Though they do not guarantee to provide optimal solutions, yet, they can obtain near optimal solutions in reasonable amount of computational time.

Search-based approach to software module clustering was first suggested by Mancoridis *et al.* [3] using hill climbing as the primary search technique. Their work aimed at the automatic recovery of the modular structure of a software system from its source code. They experimented on many systems and evaluated the results based on the feedback of system designers and found to be good. Thereafter they developed a clustering tool called Bunch [4] that creates system decomposition automatically by treating clustering as an optimization problem. Thereafter, many researchers experimented with the automatic software module clustering using many optimization techniques. Doval *et al.* [5] treated the problem of finding good clustering as an optimization problem and implemented using a Genetic Algorithm to search the extraordinarily large solution space of all possible MDG partitions. The effectiveness of the technique is demonstrated by applying it to a medium sized software system. Harman *et al.* [6] introduced a normalized representation for a software modularization, to reduce the size of the search space and to improve the results for Genetic Algorithms and also suggested a new crossover operator which is designed to promote the formation and retention of building blocks and found through their work that this crossover technique is better suited to genetic approaches than standard crossover.

Kiarash Mahdavi *et al.* [7] employed multiple hill climbing approach to software module clustering. They demonstrated that a set of multiple hill climbs can be combined to locate good 'building blocks' for subsequent searches. Building blocks are formed by identifying the common features in a selection of best hill climbs. They found that this process reduced the search space, and at the same time 'hard wired' the parts of the solution. Their empirical study shows that the multiple hill climbing approach guides the search to higher peaks in subsequent executions. Bilal Khan *et al.* [8] also treated software clustering as an optimization problem and proposed an automated technique to get near optimal decompositions of relatively independent subsystems, containing interdependent modules. They implemented using self adaptive Evolution Strategies to search a large solution space consisting of modules and their relationships and compared their proposed approach with a widely used Genetic Algorithm based approach on a number of test systems and found that the suggested approach is effective in generating quality solutions for all the test cases. Praditwong [9] experimented on real-world problems of software module clustering by metaheuristic search methods such as Genetic Algorithms and introduced the Grouping Genetic Algorithm (GGA) to the benchmarks. The empirical results reported that the GGA outperforms a Genetic Algorithm with string representation.

Recently, Praditwong *et al.* [10] formulated the software module clustering as a multi-objective search-based problem based on several objectives. They evaluated the effectiveness of the multi-objective approach on several real-world module clustering problems using a Two-Archive multi-objective evolutionary algorithm (Praditwong *et al.* [11]) and claimed that the multi-objective approach produced significantly better solutions than the existing single-objective approach.

This paper presents a new metaheuristic Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA) which is a hyper-heuristic based multi-objective evolutionary algorithm for the solution of multi-objective software module clustering problem. MHypEA uses twelve low-level heuristics based on various techniques of selection, crossover and mutation operators. The designed selection mechanism selects one of the low-level heuristics based on reinforcement learning with adaptive weights. The efficacy of the MHypEA is tested on six real-world module clustering problems taken from the literature and found that the hyper-heuristic approach has led to high quality solutions in very less computational time.

The rest of the paper is organized as follows. Section 2 describes the software module clustering problem as single and multi-objective search problems. Section 3 presents the origin of hyper heuristics along with its central idea. The proposed approach is provided in section 4. Section 5 presents the Experiments. The results obtained by the MHypEA are shown in section 6. Concluding remarks are given in section 7.

2. SOFTWARE MODULE CLUSTERING

Software module clustering is studied broadly in the literature as a single-objective search problem; but recently, it is formulated as a multi-objective search problem. This section explains both the versions.

2.1 Single-objective software module clustering problem

Software module clustering is the process of grouping the software modules into various clusters to improve program structure and to assist in easy maintenance. The input to the search algorithm is the MDG; where the nodes represent the modules and the edges represent the relationships among the modules. MDGs can be either weighted or unweighted. In a weighted MDG, the edges are assigned with weights representing the strength of relationship between the modules and an unweighted MDG depicts the relationship among modules by the presence or absence of an edge. A simple array is used as the representation scheme to map the modules to clusters. The array index represents the module and the content of the array corresponds to the cluster to which the module is assigned to. The sole objective used in any single-objective formulation of the software module clustering is the Modularization Quality (MQ) measure.

The Modularization Quantity (MQ) is defined as [10]

$$MQ = \sum_{k=1}^{n} MF_k \qquad (1)$$

where MF_k is the Modularization Factor of the kth cluster and n is the total number of clusters identified by the algorithm. Modularization Factor (MF) is the ratio of intra-edges and inter-edges in each cluster and is given by

$$MF_k = \begin{cases} 0 & if \quad i = 0 \\ \dfrac{i}{i + j/2} & if \quad i > 0 \end{cases} \qquad (2)$$

i is weight of the intra edge which has its both ends in the same cluster and j is weight of the inter edge which has its ends in two different clusters. For an unweighted MDG, the weight is considered as 1.

The primary objective of software module clustering is to produce software clusters that are of high quality with maximum intra-connectivity (cohesion) and minimum inter-connectivity (coupling). Intra-connectivity is the measure of density of connections among the modules in a single cluster. High intra-connectivity signifies a good clustering; as the largely dependent modules are grouped into the same cluster. Inter-connectivity is the measure of density of connections among modules in different clusters. Low inter-connectivity signifies a good clustering; as the clusters are highly independent of each other. The Modularization Quantity (MQ) establishes the trade-off between intra-connectivity and inter-connectivity and rewards cohesion and penalizes coupling.

2.2 Multi-objective software module clustering problem

As maximizing the cohesion and minimizing the coupling are two conflicting objectives, Praditwong *et al.* [10] reformulated the software module clustering as a multi-objective problem and formulated two multi-objective approaches – Maximizing Cluster Approach (MCA) and Equal-size Cluster Approach (ECA).

The objectives identified under MCA approach are

- maximization of sum of intra-edges of all clusters
- minimization of sum of inter-edges of all clusters
- maximization of number of clusters
- maximization of MQ
- minimization of the number of isolated clusters

The main aim of the MCA approach is to achieve good partitions having high cohesion and low coupling; while maximizing the number of clusters and minimizing the number of isolated clusters.

The ECA approach encourages producing clusters of nearly equal size, and promotes the decomposition of the software system into approximately equal size clusters. The objectives identified in this approach are

- maximization of sum of intra-edges of all clusters
- minimization of sum of inter-edges of all clusters
- maximization of number of clusters
- maximization of MQ
- minimizing the difference between minimum and maximum number of modules in a cluster

As it is a multi-objective optimization, the notion of optimality changes as these solutions are trade-offs or good compromises among the objectives [12]. In order to generate these trade-off solutions, a notion of optimality called Edgeworth-Pareto optimality is used which states that a solution to a Multi-objective problem is Pareto optimal, if there exists no other feasible solution which would improve some criterion without causing a simultaneous degrading in at least one other criterion. The use of this concept almost always gives a set of non-dominated solutions, which is called the Pareto optimal set.

3. HYPER-HEURISTICS

Hyper-heuristics are often defined as "*heuristics to choose heuristics*" [13]. A heuristic is considered as a rule-of-thumb that reduces the search required to find a solution. Meta-heuristic operates directly on the problem search space with the goal of finding optimal or near-optimal solutions; whereas the hyper-heuristic operates on the heuristics search space which consists of all the heuristics that can be used to solve a given problem. Thus, hyper-heuristics are search algorithms that do not directly solve problems, but, instead, search the space of heuristics that can then solve the problem. Therefore Hyper-heuristics are an approach that operates at a higher level of abstraction than a metaheuristic.

The term Hyper-heuristics was coined by Cowling *et al.* and described it as "*The hyper-heuristics manage the choice of which lower-level heuristic method should be applied at any given time, depending upon the characteristics of the heuristics and the region of the solution space currently under exploration*." [14]. So, they are broadly concerned with intelligently choosing a right heuristic. The main objective of hyper-heuristics is to evolve more general systems that are able to handle a wide range of problem domains. A general frame work of a hyper-heuristic is presented in Algorithm 1 [13].

Algorithm 1 : Hyper-heuristic algorithm

```
1: Start with a set H of heuristics, each of which is applicable
    to a problem state and transforms it to a new problem state.
2: Let the initial problem state be $S_0$
3: If the problem state is $S_i$ then find the heuristic that is most
    suitable to transform the problem state to $S_{i+1}$
4: If the problem is solved, stop. Otherwise go to step 3
```

4. PROPOSED APPROACH

This section describes our proposed approach. The general format of the low-level heuristics identified is EA/selection/crossover/mutation. Selection refers to the way in which the parents are selected for generating the offspring. Two types of selection are employed – *rand* and *rand-to-best*. In *rand*, both the parents are selected randomly from the population, whereas in *rand-to-best*, one parent is selected randomly from the population and the other parent is an elite (the best one). Three types of crossover operators are used to generate the offspring. The first operator is *uniform crossover*; where in the offspring is generated by randomly selecting each gene from either of the parents. The second operator is a *hybrid crossover1 (hc1)* that is defined by hybridizing the single-point crossover with uniform crossover. First a crossover point is selected and the offspring is generated by copying each gene from the parent to the offspring till the crossover point. Thereafter, the remaining genes are taken from either of the parents randomly. The third operator is a *hybrid crossover2 (hc2)* that is framed by the hybridization of two-point crossover with uniform crossover. Two crossover points are selected and the offspring is generated by taking the genes from the parent till the first crossover point. Subsequently the genes are taken randomly from either of the parents till the second crossover point. Thereafter the remaining genes are copied from the parent to the offspring. Two types of mutation are defined - *copy* and *exchange*. In the first mutation operator, two genes are selected randomly and the second gene is copied into the first one. In the second mutation operator, two randomly selected genes exchange their positions. Based on the discussed selection, crossover and mutation operators, twelve low-level heuristics are proposed for the hyper-heuristic and are shown in Table 1.

Table 1 Set of low-level heuristics used by the proposed hyper-heuristic

Group with ***copy*** mutation	Group with ***exchange*** mutation
h1 : EA/rand/uniform/copy	h7: EA/rand/uniform /exchange
h2 : EA/rand-to-best/uniform/copy	h8 : EA/rand-to-best/uniform/ exchange
h3 : EA/rand/hc1/copy	h9 : EA/rand/hc1/ exchange
h4 : EA/rand-to-best/hc1/copy	h10 : EA/rand-to-best/hc1/ exchange
h5 : EA/rand/hc2/copy	h11 : EA/rand/hc2/ exchange
h6 : EA/rand-to-best/hc2/copy	h12 : EA/rand-to-best/hc2/ exchange

The proposed hyper-heuristic selects a promising low-level heuristic in all the iterations based on the information about the effectiveness of each low-level heuristic accumulated in previous runs. This is implemented through the principle of reinforcement learning [15]. The key idea is to "*reward*" improving low-level heuristics in all the iterations of the search by increasing its weight and "*punish*" poorly performing ones by decreasing its weight. The weights of low-level heuristics are adaptively changed as the search progresses and reflect the effectiveness of low-level heuristics at any stage of the search.

Initially all the low-level heuristics are assigned with equal weight. The weight of a heuristic is changed as soon as a heuristic is called and its performance is evaluated. If the selected heuristic lead to an improvement of the objective function, its weight is increased, otherwise it is decreased. All the weights are bounded from above and from below. Thus the current values of the weights indicate the information about the past experience of using the corresponding heuristics. The roulette-wheel approach is used to select a heuristic randomly with the probability proportional to its weight [16].

The framework of the proposed hyper-heuristic is shown in Figure 1. The proposed hyper-heuristic consists of two phases. The first phase selects the type of mutation to be adopted (copy or exchange). This is done randomly with equal probability of both the groups being selected. Here, CR is a random number drawn from a uniform distribution on the unit interval, to select an EA model either with *copy* or with *exchange* mutation with equal probability. The values of h_b and h_e denotes the beginning and ending of the subscripts of low-level hyper-heuristics selected. The second phase selects a specific low-level heuristic within the selected group. This phase uses the reinforcement learning strategy with adaptive weights using roulette-wheel to select a specific model of EA. The pseudo code of the MHypEA is given in Algorithm 2.

Algorithm 2 : Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA)

```
1: Initialize parent population
2: Evaluate the fitness of parent population
3: **While** (not termination-condition) **do**
4:     Select a low-level heuristic based on the selection
       mechanism
5:     Apply the selected low-level heuristic on the parent
       population and obtain offspring population
6:     Evaluate the offspring population
7:     Combine parent and offspring populations
8:     Perform non dominated sorting on the combined
       population and select the individuals from the best
       fronts for the next iteration
9: **end while**
```

5. EXPERIMENTS

This section is devoted to the description of test problems taken from the literature and the parameters that are selected for performance evaluation.

5.1 Test problems

The performance of the MHypEA is studied on six real-world software module clustering problems taken from the literature [10]. The number of modules varies from 20 to 174 and the links among the modules from 57 to 360. All the considered MDGs are unweighted. The description of the test problems is given Table 2.

5.2 Experimental setup

The algorithmic parameters are based on the number of modules (N). The population size as well as the number of generations is fixed at 10N. Initially all the low-level heuristics are assigned with weight 1. The algorithm has been implemented in MATLAB 7.6.0, on an Intel® Core™ 2 Duo CPU T6600 @2.20 GHz processor, 3 GB RAM and Windows 7 platform.

5.3 Methodology

To assess the performance of the proposed Hyper-heuristic, we have performed 30 independent runs of the algorithm on each test problem for both the approaches (ECA and MCA) and calculated the mean and standard deviation of the three main objectives MQ, intra-edges and inter-edges. In each run, the solution with the highest MQ is chosen to be the best solution. The collected results are compared against the results of Two-Archive Multi-objective Evolutionary algorithm reported in the literature [10].

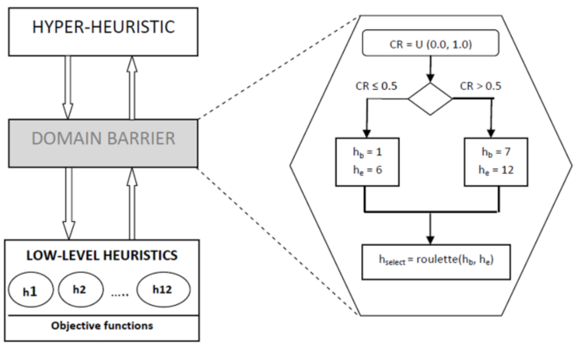

Figure 1. Framework of the proposed Hyper-heuristic

6. RESULTS AND ANALYSIS

In this section the results obtained by the MHypEA are reported. The performance of the algorithm is assessed on two factors – quality of the obtained solutions and the computational time. As the quality of clustering depends on the intra-edges (cohesion) and inter-edges (coupling), the three main objectives – MQ, intra-edges and inter-edges are considered for examining the quality of the solutions.

The values of the MQ obtained by the MCA approach using MHypEA and Two-Archive Multi-objective Evolutionary algorithm are shown in Table 3. In all the test problems, MHypEA is able to achieve highest score for MQ. Table 4 reports the values obtained by the MCA approach for the intra-edges and the inter-edges using MHypEA and Two-Archive Multi-objective Evolutionary algorithm. Interestingly, in all the test problems MHypEA is able to achieve higher values for the intra-edges and lower values for the inter-edges. From the reported results it is evident that the MHypEA outperformed in achieving high cohesion and low coupling (in other words good modular structures) in all the six test problems. And also the values of standard deviation indicate the consistent performance of the algorithm throughout.

Table 3 MQ values of the best solutions found by the MCA approach

Test problem	Two-Archive Algorithm		MHypEA	
	Mean	STD	Mean	STD
mtunis	2.294	0.013	2.310	0.011
ispell	2.269	0.043	2.334	0.027
rcs	2.145	0.034	2.204	0.028
bison	2.416	0.038	2.653	0.032
grappa	11.586	0.106	12.476	0.101
incl	11.811	0.351	13.492	0.295

Table 2 : Test Problems

Name	Modules	Links	Description
mtunis	20	57	An operating system for educational purposes written in the Turing language
ispell	24	103	Software for spelling and typographical error correction in files.
rcs	29	163	Revision Control System used to manages multiple revisions of files
bison	37	179	General-purpose parser generator for converting grammar descriptions into C programs
grappa	86	295	Genome Rearrangements Analyzer under Parsimony and other Phylogenetic Algorithms
incl	174	360	Graph drawing tool

Table 5 reports the MQ values attained by ECA approach using both the algorithms. Though, the difference of the scores of MQ values achieved by MHypEA and Two-Archive Multi-objective Evolutionary Algorithm are not high, however, MHypEA is able to achieve better values in comparison. The values obtained for the intra-edges and inter-edges by the ECA approach are shown in Table 6. As in the case of MCA approach, MHypEA is again able to attain higher intra-values and lower inter-values in all the test problems in comparison; pointing towards good modular structures.

Table 7 compares the values of MQ obtained by MCA and ECA approaches using MHypEA. The assessment between the two approaches indicates that MHypEA is able to achieve comparable values of MQ in both the cases; proving its consistency in achieving good modular structures irrespective of the approach chosen.

Coming to the second comparison criteria, the computational time is measured in terms of number of function evaluations. Table 8 depicts the number of function evaluations expended by both the algorithms. Remarkably, MHypEA is able to obtain high quality solutions with a computational time of nearly one-twentieth of the computational time spent by the Two-Archive Multi-objective Evolutionary algorithm, which clearly indicates that the hyper-heuristic approach is "fast" enough to reach out to the optimum.

Table 4 Intra-edges and Inter-edges of the best solutions found by the MCA approach

Test Problem	Two-Archive Algorithm				MHypEA			
	Intra-edges		Inter-edges		Intra-edges		Inter-edges	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
mtunis	24.633	2.092	64.733	4.185	26.333	1.516	61.333	3.032
ispell	23.100	3.220	159.800	6.440	29.667	2.581	146.667	5.163
rcs	45.133	15.335	235.733	30.669	49.500	9.306	227.000	18.612
bison	40.367	8.231	277.267	16.463	49.933	4.593	258.133	9.187
grappa	84.767	11.190	420.467	22.380	98.625	9.023	392.750	16.853
incl	91.767	14.024	536.467	28.048	138.213	11.985	444.834	23.071

Table 6 Intra-edges and Inter-edges of the best solutions found by the ECA approach

Test Problem	Two-Archive Algorithm				MHypEA			
	Intra-edges		Inter-edges		Intra-edges		Inter-edges	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD
mtunis	27	0	60	0	27	0	60	0
ispell	30.033	2.798	145.933	5.595	31.066	2.503	140.867	4.006
rcs	47.567	7.859	230.867	15.719	51.600	6.538	222.800	12.076
bison	52.800	6.217	252.400	12.434	54.100	4.279	249.800	10.559
grappa	101.167	8.301	387.667	16.601	105.600	7.861	381.800	15.722
incl	140.200	3.836	439.600	7.673	148.453	3.002	428.870	7.098

Table 5 MQ values of the best solutions found by the ECA approach

Test problem	Two-Archive Algorithm		MHypEA	
	Mean	**STD**	**Mean**	**STD**
mtunis	2.314	0.000	2.314	0.000
ispell	2.339	0.022	2.341	0.021
Rcs	2.239	0.022	2.242	0.019
bison	2.648	0.029	2.651	0.026
grappa	12.578	0.053	12.584	0.050
incl	13.511	0.059	13.518	0.052

7. CONCLUSION

This paper presents a Fast Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA) for the solution of Multi-objective software module clustering. Software module clustering is an important problem in software engineering; as good modular structures facilitates easy maintenance of the software systems. MHypEA incorporates twelve low-level heuristics which are based on different methods of selection, crossover and mutation operations of Evolutionary Algorithms. The designed selection mechanism selects a low-level heuristic based on reinforcement learning with adaptive weights. The incorporation of Hyper-heuristic, makes MHypEA fast and effective by suitably adapting itself to the dynamics of the problem by invoking a suitable heuristic for the purpose. The MHypEA is found to be quite promising in maintaining the right balance between exploration and exploitation of the search space which is a key issue in optimisation. MHypEA effectively solves the multi-objective software module clustering problem and produces good modular structures with high cohesion and low coupling. The efficacy of MHypEA for the solution of Multi-Objective software module clustering is evaluated on six real-world software module clustering problems and is compared against the Two-Archive Multi-objective Evolutionary algorithm. The comparison is based on three main objectives – MQ, intra-edges and inter-edges; along with the number of function evaluations. In all the six test problems the MHypEA produced high quality solutions with a computational time of one-twentieth of the time expended by Two-Archive Multi-objective Evolutionary algorithm.

8. ACKNOWLEDGEMENTS

The authors are extremely grateful to the Revered Prof. P.S. Satsangi, Chairman, Advisory Committee on Education, Dayalbagh for continued guidance and support. The authors are also thankful to Spiros Mancoridis and Mark Harman for providing the MDGs referenced in the paper.

Table 7 MQ values obtained by MHypEA

Test problem	MCA approach		ECA approach	
	Mean	**STD**	**Mean**	**STD**
mtunis	2.310	0.011	2.314	0.000
ispell	2.334	0.027	2.341	0.021
rcs	2.204	0.028	2.242	0.019
bison	2.653	0.032	2.651	0.026
grappa	12.476	0.101	12.584	0.050
incl	13.492	0.295	13.518	0.052

Table 8 Number of function evaluations

Test Problem	Two-Archive Algorithm	MHypEA
mtunis	800000	40200
ispell	1152000	57840
rcs	1682000	84390
bison	2738000	137270
grappa	14792000	740460
incl	60552000	3029340

9. REFERENCES

[1] Boehm, B.W. (1975). “The High Cost of Software”, in Horowitz E., Practical Strategies For Developing Large Software Systems”, Addison Wesley.

[2] Parikh, G., Zvegintzov, N. (1983) “Tutorial on Software Maintenance”, IEEE Computer Society press, Silver Spring Maryland.

[3] Spiros Mancoridis, Brian S. Mitchell, C. Rorres, Yih-Farn Chen, and Emden R. Gansner (1998) “Using automatic clustering to produce high-level system organizations of source code”. In International Workshop on Program Comprehension (IWPC’98), pages 45–53, Los Alamitos, California, USA. IEEE Computer Society Press.

[4] Spiros Mancoridis, Brian S. Mitchell, Yih-Farn Chen, and Emden R. Gansner (1999) “Bunch: A clustering tool for the recovery and maintenance of software system structures”. In Proceedings of IEEE International Conference on Software Maintenance, pages 50–59. IEEE Computer Society Press.

[5] Doval, D., Mancoridis, S., and Mitchell, B. S. (1999) “Automatic clustering of software systems using a genetic algorithm”. In International Conference on Software Tools and Engineering Practice (STEP’99), Pittsburgh, PA, 30 August - 2 September.

[6] Harman, M., Hierons, R., and Proctor, M. (2002) “A new representation and crossover operator for search-based optimization of software modularization. In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, 9-13 July 2002, Morgan Kaufmann Publishers, pp. 1351–1358.

[7] Kiarash Mahdavi, Mark Harman, and Robert Mark Hierons (2003) “A multiple hill climbing approach to software module clustering” . In IEEE International Conference on Software Maintenance, pages 315– 324,

Los Alamitos, California, USA, September 2003. IEEE Computer Society Press

[8] Bilal Khan, Shaleeza Sohail and M. Younus Javed (2008) “Evolution Strategy Based Automated Software Clustering Approach”; 2008 International Conference on Advanced Software Engineering & Its Applications (ASEA 2008) held on December 13 - 15, Hainan, China.

[9] Praditwong, K. (2011) “ Solving Software Module Clustering Problem by Evolutionary Algorithms” , Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE 2011), 11-13 May 2011, Nakhon Pathom, Thailand, pp. 154-159.

[10] Praditwong, K., Harman, M., Xin Yao (2011) “Software Module Clustering as a Multi-Objective Search Problem”, IEEE Transactions on Software Engineering, Volume 37(2), 2011, pp. 264-282.

[11] Praditwong, K., and Xin Yao (2006) “A New Multi-objective Evolutionary Optimisation Algorithm: The Two-Archive Algorithm”. In Proceedings of the 2006 International Conference on Computational Intelligence and Security, volume 1, pages 286–291, Guangzhou, China.

[12] Deb, K. (2001) “Multi-Objective Optimization using Evolutionary Algorithms”, Wiley Chichester, UK.

[13] Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S. (2003) “Handbook of metaheuristics”, chapter 16, “Hyper-heuristics: an emerging direction in modern search technology”, pp. 457–474, Kluwer Academic Publishers.

[14] Cowling, P.I., Kendall, G., Soubeiga, E. (2001) “Hyperheuristic Approach to Scheduling a Sales Summit”, Selected papers of Proceedings of the Third International Conference of Practice And Theory of Automated Timetabling, Springer LNCS vol. 2079, pp. 176-190.

[15] Kaelbling, L.P., Littman, M.L., Moore, A.W. (1996) “Reinforcement learning: a survey”, Journal of Artificial Intelligence Research 4, 237–285.

[16] Nareyek,A. (2003) “Choosing search heuristics by non-stationary reinforcement learning”, In Metaheuristics: Computer decision-making, pp. 523–544. Kluwer Academic Publishers,Dordrecht.