SCHEDULING AND INSPECTION PLANNING IN SOFTWARE DEVELOPMENT PROJECTS USING MULTI-OBJECTIVE HYPER-HEURISTIC EVOLUTIONARY ALGORITHM

A.Charan Kumari[1] and K. Srinivas[2]

[1]Department of Physics & Computer Science, Dayalbagh Educational Institute, Dayalbagh, Agra, India
charankumari@yahoo.co.in
[2]Department of Electrical Engineering, Dayalbagh Educational Institute, Dayalbagh, Agra, India
ksri12@gmail.com

ABSTRACT

This paper presents a Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA) for the solution of Scheduling and Inspection Planning in Software Development Projects. Scheduling and Inspection planning is a vital problem in software engineering whose main objective is to schedule the persons to various activities in the software development process such as coding, inspection, testing and rework in such a way that the quality of the software product is maximum and at the same time the project make span and cost of the project are minimum. The problem becomes challenging when the size of the project is huge. The MHypEA is an effective metaheuristic search technique for suggesting scheduling and inspection planning. 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 efficacy of the algorithm has been studied on randomly generated test problem.

KEYWORDS

Scheduling and Inspection planning, software project development, Multi-objective optimization, Hyper-heuristics, Evolutionary Algorithm

1. INTRODUCTION

The planning of software projects involves various intricacies such as the consideration of resources, scheduling of members, dependencies among various activities, project deadlines and many other restrictions. Besides these restrictions, the waiting times and other external uncertainties also complicate the planning process. Due to the increase in the size and complexity of the software projects, it is becoming a difficult task for the project managers to have a control on the development costs and to maintain the deadlines of the projects. These two factors are in turn dependent on the efficient scheduling of members to various activities involved in the software development process such as coding, inspection, testing etc. Thus, the main goal of scheduling and inspection planning process are to achieve high quality software product with minimum development cost and project make span.

DOI : 10.5121/ijsea.2013.4304

Search-based approaches to software project management were investigated by many researchers. In 2001, Carl K. Chang and *et al.* [1] developed a new technique based on genetic algorithms (GA) that automatically determines, using a programmable goal function, a near-optimal allocation of resources and resulting schedule that satisfies a given task structure and resource pool. In their method, they assumed that the estimated effort for each task is known a priori and can be obtained from any known estimation method such as COCOMO. Based on the results of these algorithms, the software manager will be able to assign tasks to staff in an optimal manner and predict the corresponding future status of the project, including an extensive analysis on the time and cost variations in the solution space. The results of the GA algorithm were evaluated using exhaustive search for five test cases. In these tests the proposed GA showed strong scalability and simplicity.

In 2005, Thomas Hanne and Stefan Nickel [2] modelled the problem of planning inspections and other operations within a software development (SD) project with respect to the objectives quality (no. of defects), project make span, and costs, as a multi-objective optimization problem. Their considered model of SD processes comprises the phases coding, inspection, test, and rework and includes the assignment of operations to persons and the generation of a project schedule. They developed a multi-objective evolutionary algorithm and studied the results of its application to sample problems.

E. Alba and J. F. Chicano [3], tackled the general Project Scheduling Problem with genetic algorithms. In their approach, they combined the duration and cost objectives into a single fitness function using weights in such a way that one can adjust these fitness weights to represent particular real world project. They developed an automated tool based on genetic algorithms that can be used to assign people to the project tasks in a nearly optimal way trying different configurations concerning the relative importance of the cost and duration of the project. They have performed an in depth analysis with an instance generator and solved 48 different project scenarios and performed 100 independent runs for each test to get statistically meaningful solutions. Their experiments concluded that the instances with more tasks are more difficult to solve and their solutions are more expensive and the projects with a larger number of employees are easier to tackle and can be driven to a successful end in a shorter time.

Leandro L. Minku et al. [4] presented novel theoretical insight into the performance of Evolutionary Algorithms (EA) for the Project Scheduling Problem. Their theory inspired improvements in the design of EAs, including normalisation of dedication values, a tailored mutation operator, and fitness functions with a strong gradient towards feasible solutions. According to their findings, Normalisation removes the problem of overwork and allows an EA to focus on the solution quality and facilitates finding the right balance between dedication values for different tasks and allows employees to adapt their workload whenever other tasks are started or finished. Their empirical study concludes that normalisation is very effective in improving the hit rate, the solution quality and making the EA more robust.

This paper presents a hyper-heuristic based multi-objective evolutionary algorithm [5] for the solution of scheduling and inspection planning in the software development process, based on the model suggested by Thomas Hanne and Stefan Nickel [2]. We implemented twelve low-level heuristics based on various methods of selection, crossover and mutation operators in evolutionary algorithms. The designed selection mechanism selects one of the low-level heuristics based on reinforcement learning with adaptive weights. The efficacy of the algorithm has been studied on randomly generated test problem [2]. Through our experimentation we found that MHypEA is able to explore and exploit the search space thoroughly, to find high quality

solutions. And also the proposed algorithm is able to achieve better results in half the amount of time expended by the MOEA reported in the literature.
The rest of the paper is organised as follows. Section 2 describes the scheduling and inspection planning as multi-objective search problem. The basic concepts of hyper heuristics are presented in section 3. The description of the proposed approach is provided in section 4. Section 5 presents the Experiments and the results obtained are presented and analysed in section 6. Concluding remarks are given in section 7.

2. A MULTI-OBJECTIVE SCHEDULING AND INSPECTION PLANNING PROBLEM

This section briefly describes scheduling and inspection planning as a multi-objective search-based problem as proposed by Thomas Hanne *et al.* and the detailed description can be found in [2]. The basic activities and their sequence in the model are given in Figure 1.

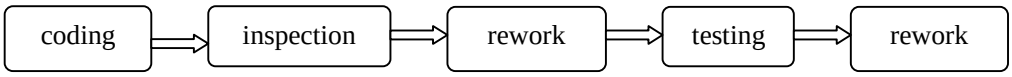

Figure 1 Sequence of activities in software development

The basic assumption of the model is that there are 'n' modules to develop, each with known size and complexity. All the activities involved in the process are done by a team of 'm' developers. The first activity in the model is coding. Each module is coded by one developer called its author. After a module is coded, it is inspected by a team of developers known as inspection team. Based on the outcome of inspection, the author reworks on the module. Thereafter, the module undergoes testing done by a developer called tester and again the author reworks on the module based on the findings of the testing process. All the activities are performed as per the sequence shown in Figure1. As per the model, inspections are assumed to be done in a pre-empting way, forcing the developers to interrupt their coding or testing process to inspect a module, as soon as it is coded by its author.

As per the assumption each module is coded by only one developer called its author. Every module is inspected by an inspection team of size 0 to m-1, with a restriction that an author of a module cannot be its inspector. Similarly, each module is tested by a tester, different from its author. For each module, priority values are assigned for coding and testing activities to determine the temporal sequence for scheduling the tasks. The three objectives [2] considered in the model are described below.

2.1. First objective – Quality of the product

The first objective is the quality of the product, measured in terms of total number of defects produced and is calculated by

$$td = \sum_{i=1}^{n} d_i \tag{1}$$

where

$$d_i = pd_{ik} - fd_{iK}^1 + rd_{iK}^1 - fd_{ik}^2 + rd_{ik}^2 \tag{2}$$

pd_{ik} denotes the produced defects during coding of an module i by an author k and is assumed to be

$$pd_{ik} = size_i.cplx_i.mdd / cqs_k \quad (3)$$

where *size* and *cplx* are the size and complexity of an module *i* and *mdd* denotes the minimum defect density and *cqs* is the coding quality skill of the author *k.*

fd^1_{iK} determines the found defects in a module *i* by the inspection team *K* and is given by

$$fd^1_{iK} = pd_i.(1 - \prod_{k \in K}(1 - itf.dds_k)) \quad (4)$$

where *itf* represents inspection technique factor and *dds* represents the defect detection skill of the inspector *k.*

It is assumed that though the rework of a module by its author removes the found defects by the inspection team, it may introduce new defects in proportional to the found defects and is given by

$$rd^1_{iK} = rdf.\ fd^1_{iK} \quad (5)$$

where *rdf* represents rework defects factor.

fd^2_{ik} denotes found defects in a module *i*, when it is tested by a tester *k*, and is given by

$$fd^2_{ik} = d'_i.(1 - e^{-dfr.tqs_k.tt_i}) \quad (6)$$

where d' represents the defects remaining in a module *i* after coding, inspection and rework, *dfr* denotes defect find rate, *tqs* is the testing quality skill of the tester *k* and *tt* represents test time of a module *i* and is determined as

$$tt_i = t_i.cplx_i.size_i \quad (7)$$

where t_i is test intensity.

Similarly, it is assumed that though the rework of a module by its author removes the found defects by the tester, it may introduce new defects in proportional to the found defects and is given by

$$rd^2_{ik} = rdf.\ fd^2_{ik} \quad (8)$$

2.2. Second objective – Project make span

In order to compute the project make span, a complete schedule for the software development process is to be made. For each module, the specific time of each activity along with its waiting times are to be calculated and the maximum time among all the modules determines the project make span. The basic assumptions made in the model are that there are no specific dependencies among the coding operations and all the inspections are carried out without any waiting times.
The coding time for a module is calculated as

$$ct_i = size_i.cplx_i/(mcp.cps_k) \quad (9)$$

where *mcp* corresponds to the maximum coding productivity and *cps* corresponds to the coding productivity skill of the author *k.*

The inspection time for a module i by the k^{th} inspector is calculated as

$$it_{ik} = size_i.cplx_i/(mip.ips_k) \quad (10)$$

where *mip* corresponds to maximum inspection productivity and *ips* corresponds to inspection productivity skill of the inspector *k*. The inspection time for a module is taken as the maximum inspection time taken among the members of the inspection team.

The rework times are calculated as

$$rt_i^1 = fd_i^1.cplx_i/(ads.mcp.cps_k) \quad (11)$$

$$rt_i^2 = fd_i^2.cplx_i/(ads.mcp.cps_k) \quad (12)$$

where rt^1_i and rt^2_i represents rework times after inspection and testing respectively and *ads* corresponds to average defect size.

The waiting time of the activities depends on the temporal sequence of the modules based on coding and testing priorities along with the availability of the developers to carry out the specific activity.

2.3. Third objective – cost

The project costs are assumed to be proportional to the effort which is measured as the total time taken for each activity. Thus the cost is calculated as

$$tc = c.\sum_i (ct_i + \sum it_{ik} + rt_i^1 + tt_i + rt_i^2) \quad (13)$$

where *c* represents unit cost of effort.

Thus, the Multi-objective scheduling and inspection planning problem is to schedule the developers to various activities of different modules, in such a way that the number of defects, project make span and cost are minimum.

3. HYPER-HEURISTICS

Hyper-heuristics are often defined as "*heuristics to choose heuristics*" [6]. 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 target 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 framework of the proposed hyper-heuristic approach is shown in Figure 2 [5].

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*" [7]. 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 [6].

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.

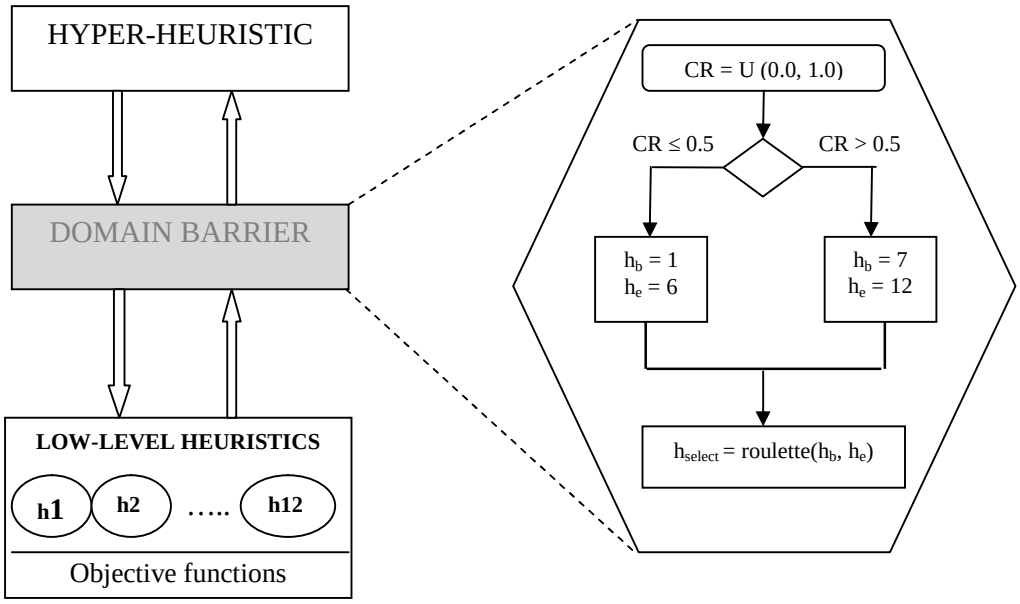

Figure 2. Framework of the proposed Hyper-**heuristic**

4. PROPOSED APPROACH

This section explains our proposed approach [5]. The designed format of the low-level heuristics is EA/selection/crossover/mutation. The selection process involves identifying the parents for generating the offspring. Two types of selection are proposed – *rand* and *rand-to-best*. In *rand*, both the parents are selected randomly from the population, while in *rand-to-best*, one parent is selected randomly from the population and the other parent is elite (the best one) selected from the set of elites. Three types of crossover operators are identified 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. The third operator is a *hybrid crossover2 (hc2)* that is framed by the hybridization of two-point crossover with uniform crossover. Two types of mutation are proposed - *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 above discussed selection, recombination and mutation operators, twelve low-level heuristics are proposed for the hyper-heuristic and are shown in Table 1 [5].

The proposed hyper-heuristic selects a capable low-level heuristic in all iterations based on the information about the efficacy of each low-level heuristic accrued during the preceding iterations. This is executed through the principle of reinforcement learning [8]. The basic idea is to "reward" improving low-level heuristics in all iterations of the search by increasing its weight gradually and "punish" poorly performing ones by decreasing its weight. The weights of low-level heuristics are changed adaptively during the search process and at any point they reflect the effectiveness of low-level heuristics.

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

In the beginning 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 called heuristic lead to an improvement in the values of the objective functions, 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 [9].

The proposed hyper-heuristic works in 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. The values of h_b and h_e denotes the subscripts of the beginning and ending of the selected low-level hyper-heuristics. The second phase explicitly selects a low-level heuristic within the selected group. This phase uses the reinforcement learning approach with adaptive weights using roulette-wheel to select a particular model of EA. The framework of the proposed hyper-heuristic is shown in Figure 2 [5]. 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.

Based on the selected low-level heuristic, an offspring population is generated and its fitness is evaluated. Thereafter, the parent and offspring populations are combined together and a non dominated sorting [10] is performed on the combined population to classify the solutions into different Pareto fronts based on the goodness of the solutions. The population for the next iteration is taken from the best fronts. The pseudo code of the MHypEA is given in Algorithm 2.

Algorithm 2 Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA) [5]

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 describes methodology applied to compute the values of objective functions, test problem considered along with the problem and algorithmic parameter settings.

5.1 Methodology

In this subsection we describe the initialization of decision variables, method adopted for crossover and mutation operations and the procedure for the evaluation of objective functions.

5.1.1 Initialization of decision variables

The decision variables taken are (author, no_inspectors, inspector, tester, coding_priority, testing_priority). Initially the developers are assigned randomly as authors to all the modules with a condition that each module is coded exactly by only one author and an author may be assigned to code more than one module. The number of inspectors for a module characterizes the inspection team size of that module which is initially assigned randomly with a value in the range 0 to (m-1), where m signifies number of developers involved in the project; the upper limit of (m-1) indicates that an author of a module cannot be its inspector and a lower limit of 0 indicates that the module is not subject to inspection. In the current scenario, the maximum number of inspectors for each module is taken as 3. The inspectors are randomly assigned for each module as dictated by the inspection team size. Similarly, a developers are assigned as testers for testing modules, with a constraint that an author of a module cannot be its tester. Further, coding_priority and testing_priority for the modules are taken as decision variables and are assigned values in the range [0, 1], to determine the temporal sequence for scheduling operations; with an intention that a module with a higher priority value is to be scheduled first than a module with a lower priority value.

5.1.2 Crossover operator

Parents are selected for crossover operation in two ways – *rand* and *rand-to-best*. Then for each module in the offspring the author, inspectors, tester, priorities of coding and testing operations are assigned with the values of a randomly chosen module from either of the parents, in accordance with the three methods discussed in section 4. In this way the offspring are generated from the parents leading to different permutations of scheduling.

5.1.3 Mutation

A simple strategy is adopted for mutation. In every offspring that is generated from the crossover operation, two modules are selected randomly and the scheduling assignment of one is copied into another in *copy* variant of mutation and the scheduling assignments are exchanged between the selected modules in the case of *exchange* variant of mutation.

5.1.4 Evaluation of objective functions

The number of defects indicating the quality of the product and the cost objectives can be calculated straightforwardly as per the formulae depicted in section 2. But the calculation of project make span is a complicated one, as one needs to frame the complete schedule of the software development process. The approach used in this paper for the computation of project make span is as follows: as a first step, the time for each activity, (depicted in the Figure 1) for each module is calculated. In the next step, the waiting times are calculated before each activity starts for each module, based on observing the given assignment of persons to modules & activities and scheduling the modules for one person according to the priorities, coding_priority for coding and rework, testing_priority for tests. Inspections are assumed to be performed as soon as the coding finishes, without any waiting time. Since inspection activity has a higher priority than other activities, it is assumed that they interrupt the coding activity of an inspector. All the activities are scheduled by considering the restrictions on their precedence. Finally the make span of each module is computed based on the actual activity times and their corresponding waiting times. The maximum make span among the modules is considered as the project make span.

5.2 Test Problem

In order to evaluate the efficiency of the proposed MHypEA, test problem and the problem parameters are taken from the literature as recommended in [2]. The following technical parameters are used:

- number of modules: n = 100
- number of developers: m = 20
- maximum coding productivity: mcp = 25 [loc/h]
- minimum defect density: mdd = 0.02 [defects/loc]
- maximum inspection productivity: mip = 175 [loc/h]
- inspection technique factor: itf = 0.45
- test intensity: ti = 0.05
- defect find rate: dfr = 0.1
- rework defects factor rdf = 0.1
- average defect size: ads = 8 [loc]
- unit costs : c = 150[EUR/h]

For the all skill attributes, a normal distribution with an expected value 0.5 and a variance of 0.1 is assumed (but ensuring that the values are in [0, 1]); with an assumption that the person on the average reach out to 50% of the optimum skill values. For the module size, a lognormal distribution with expected value 300 [loc] and variance 120 is applied. For the module complexity, a normal distribution with expected value 1 and variance 0.1 is assumed.

The population size is taken as 30 and the test problem were run for a maximum of 500 iterations. 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.

6. RESULTS

The results obtained by MHypEA on the above described test problem is presented in this section. Figures 3-5 represents the box plots of some generations visualizing the distribution of the three objective functions.

The box plots of figures 3 and 4 shows the most considerable improvement in the best values for the defects objective within the first 150 generations and costs objective within the first 250 generations of MHypEA. With respect to the duration objective there is a negligible improvement in the best values found. The conflicting nature of the objectives is evident from the three box plots.

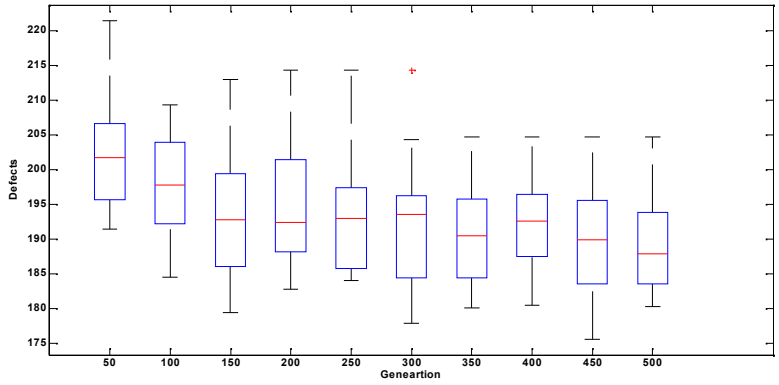

Figure 3. Defects of solutions based on generation number

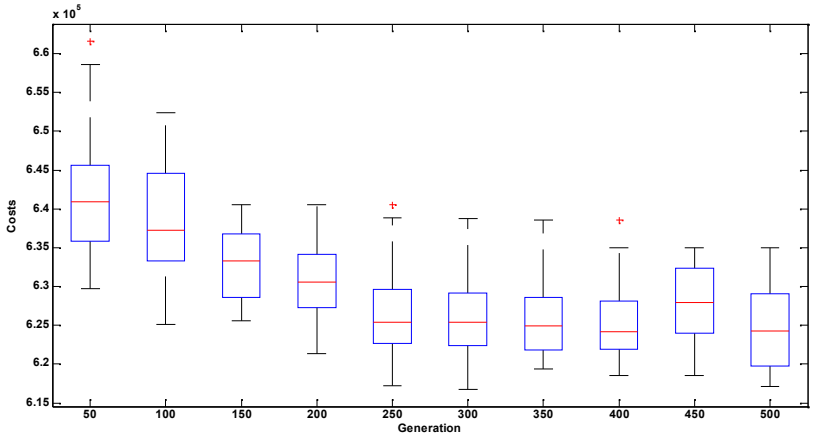

Figure 4. Costs of solutions based on generation number

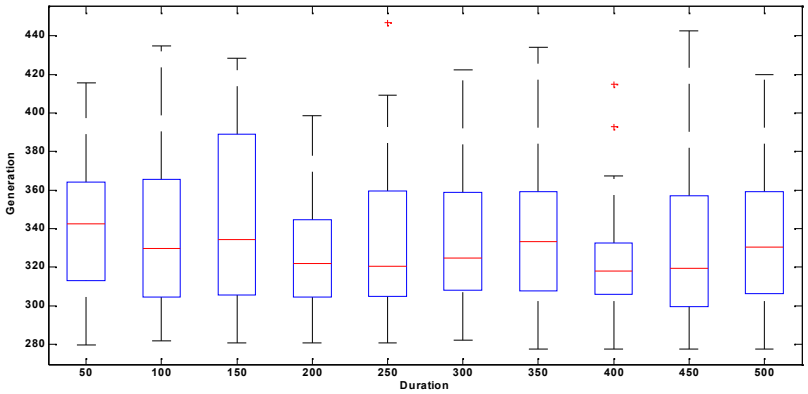

Figure 5. Duration of solutions based on generation number

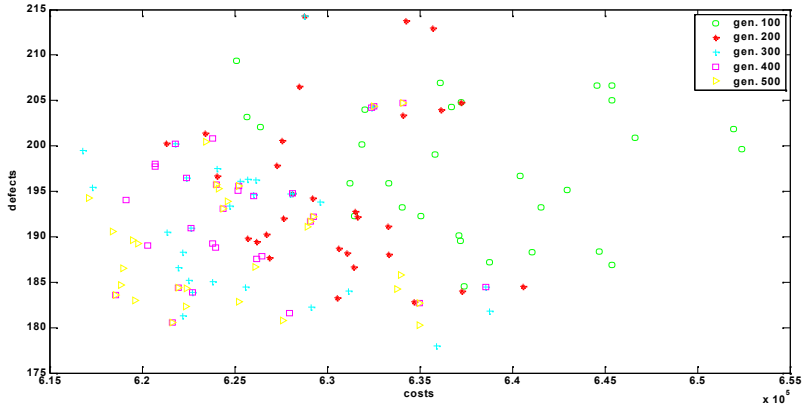

Figure 6. Costs and Defects in solutions of several generations

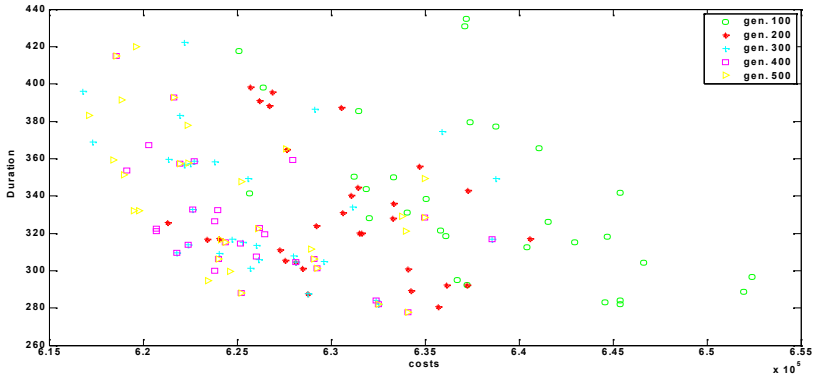

Figure 7. Costs and Duration in solutions of several generations

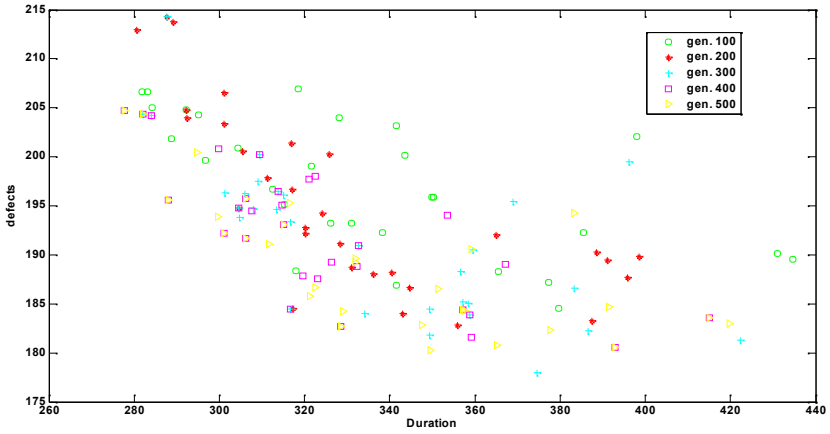

Figure 8. Duration and Defects in solutions of several generations

Figures 6-8 represents the solutions obtained by MHypEA for some selected generations for each combination of two objectives. It is apparent from the three figures 6-8 that there is a continuous progress towards the minimization of objectives and the diversity of the solutions. For example, considering the population of generation 500 there is a significant improvement in the number of defects and costs, in comparison to the population of generation 100. As the objectives are highly conflicting, in some instances one objective is decreasing only by increasing the other objective.
The comparison of the results of MHypEA with the Multi-objective Evolutionary Algorithm (MOEA) reported in the literature [2] shows that MHypEA is able to achieve better values for all the three objective functions in half the number of generations. This is perceptible from the corresponding box plots and 2-dimensional plots of MOEA [2] and MHypEA.

7. CONCLUSION

This paper presents a Multi-objective Hyper-heuristic Evolutionary Algorithm for the solution of scheduling and inspection planning in software development projects. Scheduling and inspection planning is an important problem in the software development process, as it directly influences the quality of the end product. The scheduling of personnel is considered for coding, inspection, testing and rework phases of the development process. Three highly conflicting objectives of number of defects found (indicating the quality of the product), costs and duration of the project are evaluated. A hyper-heuristic based multi-objective evolutionary algorithm is proposed for the purpose and the results are assessed. The obtained results show that the proposed algorithm is able to improve the solutions significantly with better diversity in the solutions. The comparison of the results with MOEA shows that MHypEA is able to achieve high quality solutions in half of the number of iterations.

ACKNOWLEDGEMENTS

The authors are extremely grateful to the Revered Prof. P.S. Satsangi, Chairman, Advisory Committee on Education, Dayalbagh for continued guidance and support.

REFERENCES

[1] C. K. Chang, M. J. Christensen, T. Zhang, (2001) "Genetic algorithms for project management", Annals of Software Engineering, Vol. 11, pp107-139.

[2] T. Hanne & S. Nickel, (2005) "A multiobjective evolutionary algorithm for scheduling and inspection planning in software development project", European Journal of Operational Research, Vol. 167, pp 663-678.

[3] E. Alba & J. F. Chicano, (2007) "Software project management with GAs", Information Science, Vol. 177, pp 2380-2401.

[4] Leandro L. Minku, Dirk Sudholt, Xin Yao, (2012) "Software Evolutionary Algorithms for the Project Scheduling Problem : Runtime Analysis and Improved Design", Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference, pp 1221-1228.

[5] Charan Kumari, A. & Srinivas, K., (2013) " Software Module Clustering using a Fast Multi-objective Hyper-heuristic Evolutionary Algorithm", International Journal of Applied Information Systems, Vol. 5, pp 12-18.

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

[7] Cowling, P.I., Kendall, G., Soubeiga, E. (2001) "Hyperheuristic Approach to Scheduling a Sales Summit", Proceedings of the Third International Conference of Practice And Theory of Automated Timetabling, Vol. 2079, pp 176-190.

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

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

[10] K., A. Pratap, S. Agarwal, and T. Meyarivan. (2002) "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, Vol. 6, pp 182-197.

Authors

A. Charan Kumari received the Master of Computer Applications degree from Andhra University, India in 1996 and stood first in the district, and M. Phil degree in Computer Science from Bharathidasan University, Tiruchirappalli, India. She has an excellent teaching experience of 15 years in various esteemed institutions and received various accolades including the best teacher award from the management of D. L. Reddy College, Andhra Pradesh. She is working towards the Ph.D. degree in Computer science at Dayalbagh Educational Institute, Agra, India under collaboration with IIT Delhi, India under their MoU. Her current research interests include Search-based Software engineering, Evolutionary computation and soft computing techniques. She has published papers in international conferences and journals. She is a life member of Systems Society of India (SSI).

K. Srinivas received B.E. in Computer Science & Technology, M.Tech. in Engineering Systems and Ph.D. in Evolutionary Algorithms. He is currently working as Assistant Professor in Electrical Engineering Department, Faculty of Engineering, Dayalbagh Educational Institute, Agra, India. His research interests include Soft Computing, Optimization using Metaheuristic Search Techniques, Search-based Software Engineering, Mobile Telecommunication Networks, Systems Engineering, and E-learning Technologies. He received Director's medal for securing highest marks in M.Tech. Programme at Dayalbagh Educational Institute in 1999. He is an active researcher, guiding research, published papers in journals of national and international repute, and is also involved in R&D projects. He is a member of IEEE and a life member of Systems Society of India (SSI). He is also the Publication Manager of Literary Paritantra (Systems)- An International Journal on Literature and Theory