RETRACTED CHAPTER: Software Project Planning Through Comparison of Bio-inspired Algorithms

Jesús Silva[1](✉), Noel Varela[1], Harold Neira Molina[1], and Omar Bonerge Pineda Lezama[2]

[1] Universidad de la Costa, Barranquilla, Colombia
{aviloria7,nvarela2,hneira}@cuc.edu.co
[2] Universidad Tecnológica Centroamericana (UNITEC), San Pedro Sula, Honduras
omarpineda@unitec.edu

Abstract. Currently many organizations have adopted the development of software projects with agile methodologies, particularly Scrum, which has more than 20 years of development. In these methodologies, software is developed iteratively and delivered to the client in increments called releases. In the releases, the goal is to develop system functionality that quickly adds value to the client's business. At the beginning of the project, one or more releases are planned. For solving the problem of replanning in the context of releases, a model is proposed considering the characteristics of agile development using Scrum. The results obtained show that the algorithm takes a little less than 1 min for solutions that propose replanning composed by 16 sprints, which is equivalent to 240 days of project. They show that applying a repair operator increases the hypervolume quality indicator in the resulting population.

Keywords: Genetic algorithm · Agile software projects · Multi-target

1 Introduction

During the execution of the project, it is common to note events that affect the plan, which will be called disruptive events [1]. An example of a disruptive event is an employee leaving or a new requirement being added (called user stories in the agile context) [2, 3]. The action of adjusting or updating the original plan after the occurrence of a disruptive event is known as replanning [4]. Because software development is expensive and usually has defined deadlines, project managers must immediately perform replanning to minimize economic and operational impacts and meet defined deadlines [5].

In addition, there are at least three other important objectives that must be assessed in a real project. It is desirable that, when a replanning is carried out, it does not differ

The original version of this chapter was retracted: The retraction note to this chapter is available at https://doi.org/10.1007/978-981-15-6648-6_28

© Springer Nature Singapore Pte Ltd. 2020, corrected publication 2021
N. Chaubey et al. (Eds.): COMS2 2020, CCIS 1235, pp. 340–351, 2020.
https://doi.org/10.1007/978-981-15-6648-6_27

too much from the original one, since it is considered that the initial planning is the best option for the development of the project [6]. Similarly, the history of user (HU) with the highest priority for the release must be assigned to the first sprints and thus develop the most important HU first [7]. In software development, equipment is expensive and the development time of the employees must be used in the best way to avoid wasting it [8]. Therefore, five objectives could be considered when performing replanning: time, cost, stability, release value and wasting development capacity. Therefore, the replanning problem is a multi-objective optimization problem.

This replanning problem can be seen as a generalization of the assignment problem, which is well known to belong to the NP-difficult class [9]. In addition, some of the objectives considered are in conflict, for example, if the development team seeking to minimize the delivery time increases, the project will be more expensive. For this reason, the solution proposed for this problem is based on a multi-objective genetic algorithm, which considers each objective equally important and seeks to result in a set of replanning proposals that serve as support for the project leaders.

2 Bibliographic Review

In this section we will present the most important ideas of some articles, in addition to those focused on agile methodologies, we will also take some that propose solutions to the re-planning problem.

According to the literature review so far, only [10, 11] study the re-planning problem in agile methodologies, which are only introductory and only present the characteristics of the problem, so they do not present a tool to support managers of software projects in the context of agile methodologies. They use heuristics and focus on agile methodologies. In the first [12], a model without re-planning is proposed, so it presents a static model, that is, without changes over time. They present a proposed solution to the project planning problem, the authors propose a genetic algorithm and their objectives are the time and cost of the project. They focus on the release phase, but without presenting disruptive events and planning is done for each sprint, but does not consider re-planning from an existing plan. The model seeks to assign employees to tasks. The second [13] only proposes characteristics for the project planning/re-planning model in agile methodologies since it does not present an algorithm or model for the problem. In this paper, only the objectives that they will seek to use are mentioned, which are time, cost, robustness, stability and fragmentation of HU (impact of delayed HU). Being only a proposal, it has no further detail and only mentions general disruptive events.

Most articles found take characteristics of the models developed for traditional methodologies, such as assigning employees to tasks or that planning is done only at the beginning of the project [14].

[15] considers project planning in an agile context, proposes an entire scheduling method to solve it, and only targets the time it takes to develop. We did not find any re-planning, but it presents employee assignment to tasks and does not consider an adequate estimate for HU.

The articles in which the re-planning problem is modeled emphasize that software development is a dynamic system. Changes occur during the execution phase because disruptive events occur. But, without counting the works of [16–18], in all cases they focus on traditional methodologies. They present proposals that seek to solve the problem of planning software projects, but now in a dynamic environment. In [19] we can find a model for the re-planning problem, as a solution proposal implements a genetic algorithm that seeks to minimize project time and cost. Stability is introduced as the objective of the problem and the disruptive events that it considers are two: arrival of new tasks and the movement of employees (when they arrive or leave).

As previously mentioned, the re-planning problem is current and in the last two years we can highlight three documents directed at traditional methodologies, beginning with that of [20], in which the authors seek to minimize time, cost and project stability. The disruptive events that trigger a re-planning are: the arrival or withdrawal of employees to the project, as well as when tasks arrive that were not had in the original planning. The authors seek to model team productivity and how it affects the development of their skills. As in the previous cases, the proposed solution is based on a genetic algorithm [21].

In the study by [11] we found for the first time a multi-objective function proposed in the model, in this way each objective is calculated independently. The proposal presents as a solution to a set of possible re-planning proposals. The objectives to be optimized are: project duration, development time, stability and robustness. They consider the arrival of new tasks, the withdrawal of an employee from the project and when an employee joins, as disruptive events. To validate their work, they used three real-world projects and compared them with the proposals that their algorithm gave as a solution. As in the previous case, [22] also presents a multi-objective function which is made up of: the duration of the project, the cost of development, the stability between plans, the robustness to look for an event-tolerant plan and introduce the objective of employee satisfaction when being assigned to tasks of your liking. As a solution proposal they use an algorithm composed of two heuristics. As a global solution they propose a genetic algorithm which results in a planning proposal, then they try to improve this proposal by applying an AMDE (Angle Modulated Differential Evolution) algorithm. They mention that to validate their algorithm, they were compared with real projects, in addition to cases created for testing [23].

We realize that in addition to time, cost, and stability targets, some of the items consider robustness. This objective is defined as how prone is re-planning to delay delivery dates and cost when a disruptive event occurs. That is, robustness is the ability of rethinking to cope with small changes [24].

To propose a model closer to reality, in the work of [25–27], the authors model the communication necessary between employees to carry out a task. This communication affects the project if many employees are assigned to the same task, because they spend a lot of time communicating. Finally, we find that in more current jobs the productivity and learning curve of employees is modeled, we can see that this characteristic is presented in [28, 29] and improve the model presented in [30].

As we can see, few articles talk about the re-planning problem and only three are focused on agile methodologies. None have characteristics of agile development using Scrum, since they do not use concepts such as sprint speed and story points. Therefore, there is no tool that presents proposals to solve the re-planning problem in agile methodologies.

Our work seeks to propose a re-planning model of projects developed with methodologies agile. The model that we present takes into account five objectives to optimize: time, cost, stability, use of development capacity and release value. These last two objectives are part of our contributions and are explained later.

3 The Problem of Replanning Releases in Agile Software Projects

The research proposes a model for the planning problem in agile methodologies (Scrum), especially focused on the replanning of releases. This will be called the release replanning problem in agile software projects (RPASP). After a disruptive event, the planning should be adjusted as soon as possible and with the least number of changes, so that the total cost and time of completion are not affected or, if not, that the increase is minimal, since the increase in cost must be absorbed by the company that develops it and this is translated into loss [31].

Releases are focused on delivering specific functionality and mark the delivery dates of the increments to the system. Each release is divided into one or more iterations, called sprints, as shown in the example in Fig. 1. In the release planning, the user stories needed to fulfill the objective of each release are selected and assigned to some sprint to schedule the order in which they will be developed [32]. When performing a replanning, the assignment of HU to the sprint that best suits the plan is considered, taking into account the duration, priority and dependencies of these.

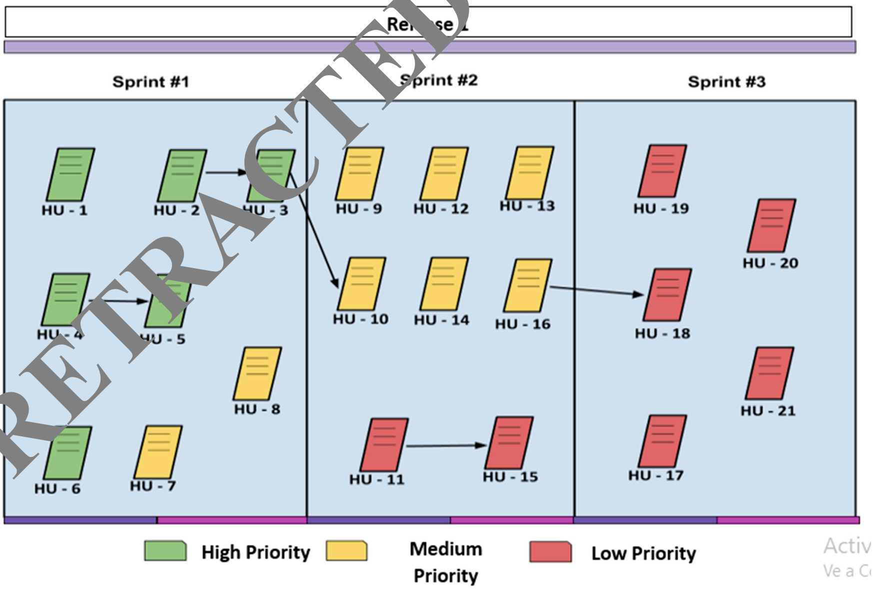

Fig. 1. Representation of an example of release planning, which consists of three sprints.

The re-planning of releases has its own characteristics, which are mainly aimed at making the scheduling of HUs more flexible and agile. Before presenting the model, it is important to introduce some concepts that are important in the agile context, and more specifically in Scrum.

- Story points. In Scrum, the estimation of the effort required to develop user stories rests with the work team. HUs are estimated with a unit called history points, which represents the effort required to develop a HU relative to a reference one. This technique is known as planning poker [33]. In this paper we consider that the team has made an effort estimate for all HUs before release is planned.
- Sprint speed. On the other hand, sprint speed is a historical metric of the ability that the team has shown to fully develop user stories in a sprint.

Our model considers the extra time that a development team can work. If a team works overtime they may have more development capacity in a sprint (sprint speed increases) [29], but the project will become more expensive, since it is necessary to pay extra time. In this work we consider 22.5% of extra work, in an 8-h day this percentage represents 2 extra hours1.

When a disruptive event occurs and a re-planning has to be made, these events are intended to have the least impact on the project, considering the following criteria.

- Cost. It is the cost of the work equipment plus the extra hours that are needed in the re-planning. To calculate it, the regular sprint speed and the sprint speed are obtained considering the extra work.
- Weather. It is the number of sprints that are necessary in the re-planning.
- Stability. Refers to the differences in user story assignment to sprints between the original release schedule and that resulting from the re-planning. It seeks to minimize this objective.
- Waste of development capacity. This target measures the sprint speed that re-planning is "wasting". We will consider as waste in the same way that extra time is used when there is still regular time available. In the projects, the development time must be used in the best way, since, as mentioned before, it is very expensive. A good re-planning should ensure that the maximum amount of time available (sprint speed) is occupied in each sprint. When adjusting the plan after a disruptive event, the new HU assignment should look for the best combination according to the sum of the history points of each sprint, so that the sprint speed in each one is occupied, preferably in its entirety.
- Release value. For the model to fit properly, in a planning or re-planning, the HUs with the highest priority must be developed in the first sprints. In a release we seek to deliver functionality that brings the greatest value to the customer's business. For example, in an online store the priority HUs are those that allow a purchase. When re-planning after a disruptive event, higher-priority HUs should be assigned to the first sprints, in order to increase the likelihood of having enough time to develop them.

4 Multi-objective Genetic Algorithm to Solve RPASP

The mentioned solution seeks to provide project leaders with a set of replanning proposals in a few minutes, since in real projects an expert usually presents a replanning proposal in at least 180 min [16], without guaranteeing that it is a good solution. By modeling the replanning problem as a multi-objective one, the result is ideally a set of solutions that present the best values found for each of the objectives and solutions that present a compensation.

This algorithm calculates the suitability of each of the replanning proposals by implementing the concept of dominance. It is said that a solution *x* dominates another solution *y* if *x* is at least as good as *y* for all the objectives and is strictly better in at least one of them. The solutions with the best aptitude are those that are not dominated by any other, this is known as non-dominance.

To know the performance of multi-objective algorithms, quality indicators are used, for example, the hypervolume, which results in a value that represents the performance of one algorithm with respect to another. The hypervolume (HV) indicator [13] measures the space covered by the non-dominated solutions with respect to a reference point and this gives some information on the convergence and diversity of individuals in the population. Therefore, one population is better than another if its HV is higher.

There are several methods to solve multi-objective optimization problems: exact, heuristic and metaheuristic. Due to the characteristics of the problem under study, we have used a metaheuristic method, in particular, a genetic algorithm (AG) based on NSGA-II [14].

Genetic algorithms are based on the principles of evolution through natural selection. Broadly speaking, a genetic algorithm considers a function to be optimized, a representation of the solution called the chromosome, genetic operators, and a function that measures the fitness of the chromosome. The search performed by an AG must have a balance between exploiting and exploring the search space. An algorithm for an AG consists of [14]:

1. Representation of the solution as a chromosome.
2. Generation of a population.
3. Repeated application of genetic operators.

For our proposed solution to the re-planning problem, NSGA-II [14] was implemented, which is a multi-objective genetic algorithm that presents elitism and conserves the diversity of the population. Each individual (chromosome) in the population represents a re-planning. NSGA-II implements a rapid classification based on non-dominance, in which all individuals in the population are compared and assigned a level. Each individual's fitness depends on their level of non-dominance, where zero represents the best fitness. The second feature of NSGA-II is an individual's stacking distance. In order to calculate this distance, individuals are increasingly arranged in each of their objectives and the distance between them is measured. This is obtained by calculating the distance that the individual has with his two closest neighbors. If the individual is at one end of the target space, then an infinite distance is assigned to him, since there are no more individuals next to him. After calculating the level of non-dominance and the stacking distance, the selection of individuals that will form part of the population in the next generation begins. As mentioned before, NSGA-II implements elitism so that generation after generation only the individuals with the best aptitude survive, that is, those with the lowest level of non-dominance [17]. The diversity in the new population depends on the stacking distance. When the selection of surviving individuals is being carried out, if two or more individuals have the same level of non-domination, then those who are in a less populated area, that is, the one with the largest stacking distance, are selected.

4.1 Proposed Multi-target Genetic Algorithm

So far, a function was defined for each of the objectives to be optimized: time, cost, stability, waste and release value. Likewise, the function that calculates the aptitude of individuals, which is based on the non-dominance of solutions [13, 14]. Next, the parts of the GA are defined: representation of the solution as chromosome, creation of the population and the genetic operators.

- Chromosome. A chromosome is a representation of the shape $a1\ a2\ a3 \cdots an$, where each position i is called a gene and ai is known as an allele. Each gene represents a HU and its value is equal to the sprint it is assigned to in the replanning. The first gene represents $h1$, the second one $h2$ and so on [15].
- Creation of the population. For the creation of the population, the first time the algorithm is executed the individuals are created from the initial planning. This means that from the initial planning a chromosome is built. The other members of the population are variations of this chromosome, which are obtained by mutating the random values of the original schedule [16].

Selection. It is done with a random binary tournament: two individuals are selected at random from the population and the one with the best aptitude wins. If they are tied in the non-dominance range, then the one with the greater stacking distance is chosen [17].

- Crossing. The crossing will be at a single point and will have a probability of 0.9. This value was selected because it presented the best preliminary results, and it is recommended by the authors of NSGA-II. It should be noted that the two-point cross was also tested, but the results were not satisfactory [18].
- Mutation. To implement this operator, each gene on the chromosome will have a probability of 0.2 of being mutated. A gene to gene mutation is implemented, where the entire chromosome is traversed and each gene has a probability between 0 and 1 at random. If it is less than or equal to 0.2, then a new sprint is assigned, different from the one that was assigned, which is also obtained at random. This value was chosen because it also gave the best results in the preliminary tests [19].

Due to the probabilistic nature of GAs, when applying one of the genetic operators to an individual it may no longer be feasible. The algorithm looks for all individuals in the population to be, so a repair operator is implemented.

- Chromosome repair. Sometimes solutions leave one or more sprinkles empty. So, although it is a valid chromosome, it is something undesirable in real projects. At this stage of repair, the chromosome is validated and if it has an empty sprint, it is removed. The procedure is as follows: The empty sprint is identified. The HU of the next sprint is run through, to the empty sprint. If there are more empty sprints, the HU are still run to avoid leaving spaces. If there are no more empty sprints, the empty sprint(s) that were run at the end of the replanning is eliminated as a result of the repair [20].

5 Results

Since there are no studies related to the replanning of projects focused on agile methodologies, there are no public test cases to compare this proposal [1, 5, 11, 15, 16]. Therefore, artificial test cases were created and the experiments were divided into two classes. The first one, with small cases, 12 and 17 HU, to test if the algorithm offers coherent solutions. The second class considers larger test cases, from 40 to 100 HU, to know if the repair operator really brings improvement to the performance of the algorithm.

5.1 Experiments with Small Test Cases

Two small test cases were created. The first one, 12 HU and 4 employees, was specially created to show the impact of the objectives. The results for this test case were consistent and small, so the impact of the objectives could be visually observed and the behavior was as expected when simulating some of the disruptive events. The results were similar to the second test case. It consists of 17 HU and 4 employees. The replanning that presented the maximum release value has an empty sprint [3].

The repair, as mentioned above, adjusts the history of users so that there are no empty sprints within the replanning. After the implementation, the results shown in Fig. 2 were obtained.

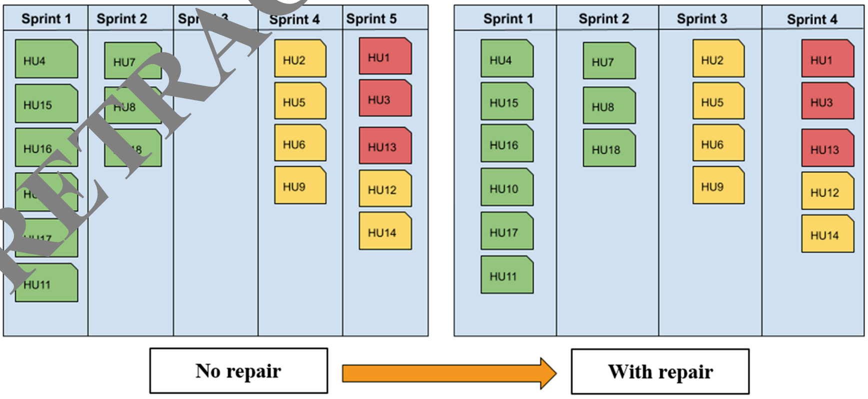

Fig. 2. Release planning after implementing empty sprint repair.

The algorithm results in a set of proposals for replanning, which present a balance in their objectives or present the best value found for any of them.

5.2 Experiments in Large Test Cases

The test cases for this phase were created with the Alba and Chicano test case generator [12], which was configured to randomly create plans with 40, 70 and 100 HU. For the number of employees, 4, 5 and 6 were considered, respectively. By running the test case generator, plans with up to 16 sprints were obtained. In each of the tests, two disruptive events were modeled: one employee leaves and a new HU is added, as these have the greatest negative impact on the planning [19, 20].

To check if the empty sprint repair really contributes to the performance of the algorithm, a test case with a number of HU and employees is obtained with the generator. Then, a disruptive event is simulated and the algorithm is run 10 times. In each execution, the same amount of HU is obtained, the same number of employees, the same disruptive event and the sprint repair is applied. From the 10 repetitions, an average of the algorithm's run time and the maximum values of all the runs for each of the targets are obtained. This procedure is repeated again, but now without the sprint repair.

Now there are 10 repeats with repair and 10 without it, besides two sets of maximum values. From the two sets, a new comparison is made again to get the overall maximums of the 20 runs. The global maximum are the reference points with which the hypervolume of each resultant population is calculated. Finally, the average hypervolume of the 10 populations with repair and the average of the 10 without repair were calculated. The summary of the results of the experiments is shown in Table 1. In the first column is the number of HUs, the second column shows the number of employees and the third column shows the disruptive event: A indicates that an employee leaves and B represents that a new HU is added. The last four columns represent, respectively, the average execution time in seconds (t) and the average of the hypervolume for the algorithm without repair and with repair.

Table 1. Data of test cases, disruptive event and summary of results obtained with the multi-target genetic algorithm, without and with repair operator.

No. HU	No. Employee	Event	No repair		With repair	
			t	HV	t	HV
40	2	A	112.352	168524.12	110.147	192541.65
40	3	B	104.100	112654.35	135.50	135201.42
70	3	A	245.20	265275.47	212.62	320354.24
70	4	B	262.70	223014.69	252.54	295241.85
100	4	A	310.54	342315.01	336.25	330124.32
100	5	B	306.24	335201.05	342.47	425842.01

The results show that the empty sprint repair helped the algorithm reach a higher hypervolume. It can be then interpreted that, when empty sprint repair is applied, the replanning of the result population shows a greater diversity and better results in the target functions. That is, they find better replanning, since, in general, the population converges to better results for the objectives of time, cost, stability, waste and release value. It can be noted that, in most cases, the average execution time increases when the repair is applied, but this is not considerable.

In fact, for the first and fourth experiments, the average execution time with the repair is less than without it. The results show that applying the repair in the algorithm results in a population with a higher HV, therefore, the algorithm has a better performance and does not significantly affect the runtime.

6 Conclusions

This paper presents the replanning of agile software project releases as a multi-objective optimization problem. The literature review indicated that there is not much work on optimization problems in agile software development methodologies. Furthermore, a single one that develops its characteristics was not found. So, as a first contribution, the proposal of a model for the problem of replanning releases of agile software projects is presented. Unlike the studies found in the literature review, this model presents specific characteristics of agile development such as: the concepts of history points and sprint speed.

The study introduces the objectives of waste and release value, besides implementing the extra time of development of a team as a configurable parameter, since each team is different.

The problem of re-planning of releases can be considered as a generalization of the allocation problem, which is a problem of the NP-difficult class. Therefore, to solve it, a multi-objective genetic algorithm that implements NSGA-II is proposed. Being a population-based algorithm, it covers the need to quickly obtain release re-planning proposals after a disruptive event occurs. In the results we can see that our algorithm presents a set of proposals to re-plan releases in less than 7 min for the most difficult cases that were tested. The analysis of the results shows that when applying the repair in all cases the HV hypervolume increases and therefore the algorithm has a better performance without hardly affecting the execution time. These solutions present 100 HU, covering 15 or 16 sprints, which we can compare with approximately 240 days of project development. In real projects, an expert generally presents a re-planning proposal in at least 180 min [7], without guaranteeing that it is a good choice.

As future work it is proposed to obtain data from real projects and with that data run the algorithm, simulate disruptive events and verify that solutions are feasible in real projects. Likewise, it will seek to implement other types of heuristics, such as MOEA / D and SMS-EMOA. It will seek to implement a second heuristic to improve the result population (local optimization). Other quality indicators that are important to our results will be investigated and implemented. Finally, it will seek to generate a support tool for real projects.

References

1. Semenkina, O.E., Popov, E.A., Ryzhikov, I.S.: Hierarchical scheduling problem in the field of manufacturing operational planning. In: IOP Conference Series: Materials Science and Engineering, vol. 537, no. 3, p. 032001. IOP Publishing (2019)

2. Phanden, R.K., Jain, A., Davim, J.P. (eds.): Integration of Process Planning and Scheduling: Approaches and Algorithms. CRC Press, Boca Raton (2019)
3. Jahr, M.: A hybrid approach to quantitative software project scheduling within agile frameworks. Project Manage. J. **45**(3), 35–45 (2014)
4. Roque, L., Araújo, A.A., Dantas, A., Saraiva, R., Souza, J.: Human resource allocation in agile software projects based on task similarities. In: Sarro, F., Deb, K. (eds.) SSBSE 2016. LNCS, vol. 9962, pp. 291–297. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47106-8_25
5. Varas, J.M., et al.: MAXCMAS project: autonomous COLREGs compliant ship navigation. In: Proceedings of the 16th Conference on Computer Applications and Information Technology in the Maritime Industries (COMPIT) 2017, pp. 454–464 (2017)
6. Ge, Y.: Software project rescheduling with genetic algorithms. In: 2009 International Conference on Artificial Intelligence and Computational Intelligence, vol. 1, pp. 439–443. IEEE, Shanghai (2009)
7. Ge, Y., Xu, B.: Dynamic staffing and rescheduling in software project management: a hybrid approach. PLoS ONE **11**(6), e0157104 (2016)
8. Shen, X., Minku, L.L., Bahsoon, R., Yao, X.: Dynamic software project scheduling through a proactive-rescheduling method. Trans. Softw. Eng. **42**(7), 658–686 (2016)
9. Shen, X.N., Minku, L.L., Marturi, N., Guo, Y.N., Han, Y.: A Q-learning-based memetic algorithm for multi-objective dynamic software project scheduling. Inf. Sci. **428**, 1–29 (2018)
10. Song, Y.J., Zhang, Z.S., Song, B.Y., Chen, Y.W.: Improved genetic algorithm with local search for satellite range scheduling system and its application in environmental monitoring. Sustain. Comput. Inf. Syst. **21**, 19–27 (2019)
11. Moosavi, S.H.S., Bardsiri, V.K.: Satin bowerbird optimizer: a new optimization algorithm to optimize ANFIS for software development effort estimation. Eng. Appl. Artif. Intell. **60**, 1–15 (2017)
12. Zheng, Z., Guo, J., Gill, E.: Swarm satellite mission scheduling & planning using hybrid dynamic mutation genetic algorithm. Acta Astronaut. **137**, 243–253 (2017)
13. Viloria, A., Acuña, G.C., Franco, D.J.A., Hernández-Palma, H., Fuentes, J.P., Rambal, E.P.: Integration of data mining techniques to PostgreSQL database manager system. Procedia Comput. Sci. **155**, 575–580 (2019)
14. Deng, M., et al.: A two-phase coordinated planning approach for heterogeneous earth-observation resources to monitor area targets. IEEE Trans. Syst. Man Cybern. Syst. (2020)
15. Ghoddousi, P., Ansari, R., Makui, A.: An improved robust buffer allocation method for the project scheduling problem. Eng. Optim. **49**(4), 718–731 (2017)
16. Tomori, H., Hiyoshi, K.: Control of pneumatic artificial muscles using local cyclic inputs and genetic algorithm. Actuators **7**(3), 36 (2018)
17. Ibraigheeth, M., Fadzli, S.A.: Core factors for software projects success. JOIV Int. J. Inf. Visual. **3**(1), 69–74 (2019)
18. da Silva Arantes, J., da Silva Arantes, M., Toledo, C.F.M., Júnior, O.T., Williams, B.C.: An embedded system architecture based on genetic algorithms for mission and safety planning with UAV. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1049–1056 (2017)
19. Perez, R., Vásquez, C., Viloria, A.: An intelligent strategy for faults location in distribution networks with distributed generation. J. Intell. Fuzzy Syst. **36**(2), 1627–1637 (2019)
20. Viloria, A., Robayo, P.V.: Virtual network level of application composed IP networks connected with systems-(NETS Peer-to-Peer). Indian J. Sci. Technol. **9**, 46 (2016)
21. Plice, L., Lau, B., Pisanich, G., Young, L.A.: Biologically inspired behavioral strategies for autonomous aerial explorers on Mars. In: 2003 IEEE Aerospace Conference Proceedings (Cat. No. 03TH8652), vol. 1, pp. 1–304. IEEE (2003)

22. Barbagallo, D., Di Nitto, E., Dubois, D.J., Mirandola, R.: A bio-inspired algorithm for energy optimization in a self-organizing data center. In: Weyns, D., Malek, S., de Lemos, R., Andersson, J. (eds.) SOAR 2009. LNCS, vol. 6090, pp. 127–151. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14412-7_7
23. Srivastava, P.R., Varshney, A., Nama, P., Yang, X.S.: Software test effort estimation: a model based on cuckoo search. Int. J. Bio Inspired Comput. **4**(5), 278–285 (2012)
24. Sheta, A.F., Ayesh, A., Rine, D.: Evaluating software cost estimation models using particle swarm optimisation and fuzzy logic for NASA projects: a comparative study. Int. J. Bio Inspired Comput. **2**(6), 365–373 (2010)
25. Tempesti, G.: Architectures and design methodologies for bio-inspired computing machines. In: SNF Professorship Application Research Plan (2003)
26. Chiang, H.S., Sangaiah, A.K., Chen, M.Y., Liu, J.Y.: A novel artificial bee colony optimization algorithm with SVM for bio-inspired software-defined networking. Int. J. Parallel Prog. 1–19 (2018)
27. Camacho, D., et al.: From ephemeral computing to deep bioinspired algorithms: new trends and applications. Future Gener. Comput. Syst. **88**, 735–746 (2018)
28. Chis, M.: Introduction: a survey of the evolutionary computation techniques for software engineering. In: Evolutionary Computation and Optimization Algorithms in Software Engineering: Applications and Techniques, pp. 1–12. IGI Global (2010)
29. Wang, L., Shen, J.: Towards bio-inspired cost minimisation for data-intensive service provision. In: 2012 IEEE First International Conference on Services Economics, pp. 16–23. IEEE (2012)
30. Wang, J., Cao, J., Li, B., Lee, S., Sherratt, R.S.: Bio-inspired ant colony optimization based clustering algorithm with mobile sinks for applications in consumer home automation networks. IEEE Trans. Consum. Electron. **61**(4), 438–444 (2015)
31. Chis, M., (ed.) Evolutionary Computation and Optimization Algorithms in Software Engineering: Applications and Techniques: Applications and Techniques. IGI Global (2010)
32. Sharma, T.K.: Estimating software reliability growth model parameters using opposition-based shuffled frog-leaping algorithm. In: Ray, K., Pant, M., Bandyopadhyay, A. (eds.) Soft Computing Applications, pp. 149–164. Springer, Singapore (2018)
33. Barocio, E., Regalado, J., Cuevas, E., Uribe, F., Zúñiga, P., Torres, P.J.R.: Modified bio-inspired optimisation algorithm with a centroid decision making approach for solving a multi-objective optimal power flow problem. IET Gener. Transm. Distrib. **11**(4), 1012–1022 (2017)