Optimizing SVM Hyperparameters using Predatory Swarms Algorithms for Use Case Points Estimation

Ardiansyah
Department of Electrical Engineering and Information Technology
Universitas Gadjah Mada
Yogyakarta, Indonesia
ardiansyah2018@mail.ugm.ac.id

Department of Informatics, Faculty of Industrial Technology, Universitas Ahmad Dahlan Yogyakarta, Indonesia
ardiansyah@tif.uad.ac.id

Ridi Ferdiana
Department of Electrical Engineering and Information Technology
Universitas Gadjah Mada
Yogyakarta, Indonesia
ridi@ugm.ac.id

Adhistya Erna Permanasari
Department of Electrical Engineering and Information Technology
Universitas Gadjah Mada
Yogyakarta, Indonesia
adhistya@ugm.ac.id

Abstract— **The productivity factor is one of the key cost drivers in use case points estimation. However, this factor is uncertain because it is obtained from experts' guesses and does not depend on reliable historical data. This study aims to optimize the hyperparameters of the support vector machine technique through the use of three predatory-inspired algorithms, namely, Komodo *mlipir* algorithm, Grey wolf optimization, and Reptile search algorithm in predicting productivity factor rate. The optimizers searched for the best hyperparameter value to get an optimum hyperplane. In this work, we developed a hybrid model that consists of three primary stages: cluster generation using bisecting k-Medoids, prediction using an optimized support vector machine, and estimation using multiple linear regression. The hybrid model was evaluated using industrial and student projects datasets. Detailed investigations demonstrated that the komodo *mlipir* algorithm yielded the best mean value of 365.3 and 739.88, respectively, which has a significantly better hyperparameter of support vector machine model in predicting productivity factors among the reptile search algorithm for datasets 4 and 5 as of 0.03 ($p < 0.05$), but there is no significant difference among the grey wolf optimizer ($p > 0.05$). Meanwhile, the grey wolf optimizer obtained the best mean value of 366.47, with a significantly better hyperparameter compared with the reptile search algorithm for dataset four as of 0.047 ($p < 0.05$).**

Keywords—effort estimation, productivity factor, support vector machine, optimization, use case points

I. Introduction

Use Case Points (UCP) is a prominent algorithmic software effort estimation framework supporting the planning phase of the software development life cycle. UCP estimates an effort by multiplying the software project size with the productivity factor (PF). PF reflects the productivity rate of the team to finish a project. There are several approaches to determining PF value: using the fixed number, which is equal to 20 person-hours (PH) per 1 UCP [1], three-level PF [2], linear regression [3], [4], and machine learning [5]. UCP uses 20 PH or three-level PF values when the historical dataset is absent.

In contrast, regression and machine learning is used when the historical dataset is available. Using a fixed number of PF has uncertainty issues because it is obtained from expert guessing and does not rely on the historical dataset. Several studies [2], [6] have demonstrated that PF is highly influenced by the environmental factor, which is part of the complexity factor in UCP. Moreover, [5] successfully predicts the PF value using class decomposition techniques [7]. Hence, we can further use the prediction model of machine learning and data mining to predict the productivity factor.

A support vector machine (SVM) is a well-known prediction or classifier algorithm. SVM is suitable for generalization and has numerous implementations in different fields, such as image processing, disease diagnosis, hydrology, and signal classification. However, SVM heavily relies on the quality of hyper-parameter value, which is the penalty parameter (C), and the kernel function gamma (γ) [8], [9]. The proper hyper-parameter values will increase prediction and estimation performance because they lead to the optimum hyperplane.

There are many examples of metaheuristic optimizers such as particle swarm optimization (PSO), genetic algorithm (GA), bat algorithm (BA), whale optimization algorithm (WOA), moth optimization algorithm (MOA), komodo *mlipir* algorithm (KMA), grey wolf optimizer (GWO), and reptile search algorithms (RSA). Between them, KMA, GWO, and RSA are grouped as predatory-based algorithms. This algorithm has some benefits, such as small population size, high scalability, effective exploration, and handling various constraints problems.

Metaheuristic optimization algorithms are widely used to tune the SVM hyperparameters. Several studies have demonstrated the application of this hybridization, such as particle swarm optimization (PSO) [10] and evolutionary competitive swarm optimization (ECSO) [11] for medical disease diagnosis, moth flame optimization (MFO) for forecasting the tunnel boring machine (TBM) advance rate (AR) [12], grey wolf optimizer (GWO) for phishing website detection [13], Harris Hawks optimization (HHO) for chemical descriptor selection [14], and social ski driver (SSD) algorithm for the classification of imbalanced data [15]. These hybrid methods show that optimizing the hyperparameter of the SVM is yielded competitive results.

Despite the promising results, however, this kind of hybrid approach in software effort estimation studies is still absent to the best of our knowledge. Therefore, based on the argument above, this study aims to optimize the hyperparameter of the support vector machine to get optimum hyperplane and improve the accuracy rate of the UCP effort estimation. The contribution of this study is to be the first empirical study using predatory-inspired algorithms for optimizing SVM hyperparameters in the software effort estimation field.

The remainder of this paper is constructed as follows: Section 2 describes the theoretical foundation of this research; Section 3 specifies the material and proposed methods; Section 4 presents and discusses the experimental results, and Section 5 describes the conclusion and future work recommendation of this study.

II. Theoretical Background

There are three primary trends in the study of use case points effort estimation: modification of the UCP sizing technique, simplifying and examining the UCP, and hybridizing the UCP with machine learning and data mining techniques. Several studies proposed the reconstruction of the UCP sizing technique. In [16] modified the use case complexity weight using fuzzy theory, while [17] successfully optimized this modified weight. In [18] added two new variables, the size-transactions and entity objects computed from the use case description. The study of [19] has modified the complexity assessment of actors and handled the non-functional requirements. This study made an essential contribution to the adaptability of the UCP for incremental development. Ref [20], [21] examined and simplified the UCP to understand the impacts of technical and environmental complexity factors. The authors suggested that adjusting the environmental factors based on the type of organization will improve the estimation precision. Whereas [22], [23] excluded several parts of UCP to simplify the calculation process of the UCP. The investigator claimed that these parts are insignificant concerning the effort estimation. Recently, [24] optimized the correction factors (ECF and TCF) and multiple regression models to improve the estimation accuracy of the modified UCP. The utilization of machine learning and data mining techniques to improve UCP performance has been studied in recent years. In [4] built cooperation between effort, UCP, and productivity by introducing a log-linear regression model. This study was then followed by a hybrid model which predicts productivity factor and effort estimation from historical data at the same time [5].

Meanwhile, [25] estimated an effort based on UCP and team productivity using the Treebost model. Indeed, we can scrutinize that none of the above studies that simultaneously predict PF and estimate an effort using a hybrid model has tried to improve the quality of the SVM hyperparameter. This study improved the work of [5] by introducing SVM hyperparameter values using predatory swarm-inspired optimization algorithms.

A. Use Case Points Estimation

Gustav Karner introduced the UCP estimation model in 1993 to calculate the size of the object-oriented-based software project [1]. The UCP converted use case specification and diagram elements in the UML documentation into a project size. A well-defined procedure should be taken to achieve high-quality size metrics as follows. First, weighting the actor elements in the use case diagram by classifying them into three-level categories: simple, average, and complex. Second, computes Unadjusted Actor Weighting (UAW) using Eq. (1).

$$UAW = \sum_{i=1}^{3} weight\ of\ actor_i * actor_i \quad (1)$$

where $weight\ of\ actor_i$ is the weight factor classified as simple = 1, average = 2, and complex = 3, while, $actor_i$ is the number of actors classified based on their category from use case diagrams. Third, based on the number of transactions in use case specification, weight the use cases by classifying them into one of three classes: simple, average, and complex. The notion of the transaction is introduced by [26], [27], which is defined as an interaction between the actor and the system indicated by a stimulus and response. Fourth, calculate Unadjusted Use Case Weighting (UUCW) using Eq. (2).

$$UUCW = \sum_{i=1}^{3} weight\ of\ use\ case_i * use\ case_i \quad (2)$$

where $weight\ of\ use\ case_i$ is the weight factor classified as simple = 5, average = 10, and complex = 15, and $use\ case_i$ is the number of use case that has been classified based on their transactions from use case specifications. Fifth, compute Unadjusted Use Case Points (UUCP) by summation of UAW and UUCW (see Eq. (3)).

$$UUCP = UAW + UUCW \quad (3)$$

Sixth, define Technical Complexity Factor (TCF) value and Environmental Complexity Factor (ECF). TCF is the factor that contributes a significant impact on project performance. TCF is computed by summation of 13 technical factors ($F_1, F_2, \ldots, F_{13}$) classified by the estimator using 5 Likert scales, as notated by Eq. (4).

$$TCF = 0.6 + (0.01 * \sum_{i=1}^{13} Tf_i * Scale_i) \quad (4)$$

where Tf_i is the technical factor weight, and $Scale_i$ is the score from 5 Likert scales. Meanwhile, ECF dramatically impacts the productivity of the project. TCF is calculated by summating 8 environmental factors ($E_1, E_2, \ldots, E_8$) classified by the estimator using 5 Likert scales, as shown in Eq. (5).

$$ECF = 1.4 + (-0.03 * \sum_{i=1}^{8} Ef_i * Scale_i) \quad (5)$$

where Ef_i is the environmental factor weight and $Scale_i$ is the score from 5 Likert scales. Finally, the software size is computed by multiplying UUCP, TCF, and ECF formulated in Eq. (6).

$$size = UUCP * TCF * ECF \quad (6)$$

Finally, the estimated effort is computed by multiplying the size with the productivity factor (PF). The standard PF of the Karner model is 20.

B. Support Vector Machine

SVM is one of the binary classifier algorithms that separated hyperplane as its essential characteristics. The basic notion of SVM is that given training data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. The input space pattern is $x \in \Re^M$, where M represents the number of input feature or dimensional and the output variable is $y \in \Re$. The data points are separated into two classes with a maximal margin by the

SVM. Margin is the maximum width of the slab parallel to the hyperplane with no interior data points. The hyperplane is formulated as $y = w^T X + b$, where X is the input vector, w is a vector, and b is a scalar. SVM used hyper-parameter C and gamma (γ). C is a parameter that defines to what extent each misclassification is penalized. Hence, the SVM model as in Eq. (7) solves the following problem:

$$\min_{\omega,b,\xi} \frac{1}{2} \|\omega\|^2 + C \sum_{i=1}^{n} \xi_i$$

$$\text{Subject to } y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1, \dots, n \tag{7}$$

$$\xi_i \geq 0, \quad i = 1, \dots, n$$

where ξ_i is a slack variable for regularization to prevent the hyperplane from overfitting the dataset. There are four kernel function parameters: polynomial, radial basis, Gaussian, and linear. Kernel functions transpose data to higher dimensions. This study uses the radial basis kernel function due to its accuracy and reliable performance as formulated in Eq. (8).

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \tag{8}$$

III. METHODOLOGY AND EXPERIMENTAL SETUP

A. Dataset preparation

An appropriate number of projects is essential for comparison with previous studies. Thus, we employed five datasets that have been common for effort estimation studies, especially for the UCP model. All datasets comprise ten features: actual effort, size, E1, E2, E3, E4, E5, E6, E7, and E8. Forty-five industrial and 65 educational projects are assigned as the first (DS1) and the second (DS2) datasets, respectively. The third dataset (DS3) is generated from the merger of DS1 and DS2 plus ten additional data. As a result, DS3 contains 120 data in total. In [28] argued that the structure of DS1 and DS2 was the same and enabled us to scrutinize the benefit of the proposed model over the heterogeneous dataset. The fourth (DS4) and fifth dataset (DS5) are generated from the selection of each data point based on their size as ruled by Eq. (9). Hence, DS4 and DS5 contain 61 and 59 data points, respectively.

$$project\ size = \begin{cases} small, if\ 100 > UCP \\ medium, if\ 100 \leq UCP < 300 \\ large, if\ 300 < UCP \end{cases} \tag{9}$$

B. Cluster generation

Bisecting k-Medoids is a clustering method that applies a basic k-medoids algorithm by splitting each cluster into two sub-clusters to construct a binary tree of clusters [5]. Algorithm 1 defines the procedure of bisecting k-Medoids. First, set a dataset as the initial cluster. Second, bisects each cluster into two coherent clusters. Third, compute the variance of clusters by Eq. (10).

$$variance = \frac{1}{N} \sum_{j=1, y_j \in C_i}^{N} \|x_j - v_i\|^2 \tag{10}$$

where N is the number of data points in the dataset, x_j is the jth data point, v_i is the center of the ith cluster (C_i), and $\|.\|$ is the Euclidean distance norm.

A cluster that has a more minor variance shows a high homogeneity. The k-Medoids procedure stops bisecting when the variance of the parent cluster is smaller than the largest variance of both child clusters. Contrary, the clustering algorithm continues to bisect.

Algorithm 1. Bisecting k-Medoids

```
(1) Input: DS1, DS2, DS3, DS4, DS5
(2) Output: The set of N clusters S={C1,C2,C3,…,CN}
(3) Initialization: V=X, S={}, nextLevel={}
(4) while size(V)>0 do
(5)    foreach cluster C in V
(6)      comp := variance(C) by Eq. (8)
(7)      [C1,C2] := k-Medoids(C,2)
(8)      comp1 := variance(C1) by Eq. (8)
(9)      comp2 := variance(C2) by Eq. (8)
(10)     if(max(comp1,comp2)<comp)
(11)       nextLevel := nextLevel U {C1,C2}
(12)     else
(13)       S := S U {C}
(14)     end if
(15)   end foreach
(16)   V := nextLevel
(17)   nextLevel := {}
(18)end while
```

C. Parameter settings

The kernel function is an essential hyperparameter in SVM. It was able to solve the problems with too many dimensions. There are four general kernel functions: Gaussian, radial basis function (RBF), polynomial, and the sigmoid. The Gaussian and RBF have good generalization capability for numerous kinds of datasets and are popular for practical use [29]. Thus, in this work, we utilize the RBF as the kernel function. We follow the variable range of hyper-parameter values [0.01, 100] for C, and [0.01, 50] for γ as introduced in [30]. The complete parameter settings for all optimizers are described in Table I.

TABLE I. PARAMETER SETTINGS

Algorithms	**Parameters**	**Ref.**
SVM+KMA	n_1: 5, n_2: 200, $n2_{Min}$: 20, $n2_{Max}$: 200, p_1:0.5, p_2: 0.5, d_1: $\frac{(m-1)}{m}$, d_2: 0.5, iteration: 10, Kernel: RBF	[31]
SVM+GWO	population: 10, iteration: 10, Kernel: RBF, *a* was linearly decreased from 2 to 0	[32]
SVM+RSA	Crocodiles: 10, α: 0.1, β: 0.1, iteration: 10, Kernel: RBF	[33]

D. Evaluation criteria

The result of the estimation model must be evaluated by a reliable evaluation measure. In this regard, we use Mean Absolute Error (MAE), see Eq. (11).

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |\hat{Y}_i - Y_i| \tag{11}$$

where N is the number of data points, $\hat{Y}_i$ is the estimated effort, and Y_i is the actual effort. This accuracy measure is also assigned as the objective function for the optimization process to get the minimum MAE.

E. Proposed Method

The proposed method consists of three main stages: cluster generation, optimization of productivity prediction, and effort estimation. Algorithm 1 is employed to generate clusters. For each run, the algorithm generates different clusters. To predict the productivity, SVM accepts the test set, which consists of eight environmental complexity factors (ECF) as input, and productivity factor value as the output or predicted PF. It is important to note that the PF value is from the medoid of a predicted cluster. PF value is a result of the division of actual effort and size. The optimizer algorithms search for the best hyperparameter value to minimize the objective function formulated in Eq. (11). The combination of SVM and predatory optimizers forms the hybrid method. Details of each algorithm can be found at [32] for GWO, [31] for KMA, and [33] for RSA.

Next, the estimated effort is calculated using multiple linear regression based on the matrix approach as notated in Eq. (12).

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + v_1 \quad (12)$$

where $\hat{Y}$ is estimated effort as a dependent variable, X_1 and X_2 is independent variable, size and predicted PF, respectively. Meanwhile, β_0 is a constant, β_1 and β_2 is the coefficient parameter of regression. The experiments are equipped with a computer Core-i7-8550U, 1.80Ghz CPU, 16 GB RAM, and Microsoft Windows 10 64-bit. Python 9.3 with Sklearn library is used to build a prediction model, and PHP 8.0.11 to develop the optimizer algorithms. The source code of the experiment can be found at https://bit.ly/39V7UZu.

IV. Results and Discussions

This study utilizes thirteen well-known mathematical functions for optimization algorithms to evaluate the proposed method [34], [35]. The first seven functions are unimodal, and the rests are multimodal functions. Unimodal functions are suitable to assess optimizers' exploitation capability because their characteristics only have one peak or global optimum. Meanwhile, multimodal functions are qualified to evaluate optimizers' exploration and local optimum avoidance capability.

Table II shows the results of unimodal benchmark functions. We can observe that GWO outperforms the KMA and RSA algorithm in most cases. In other words, GWO yields the best result in six test functions of F1, F3, F4, F5, F6, and F7. At the same time, RSA gained the best result only in the test function of F2. At the same time, Table III shows the results of multimodal benchmark functions. We can observe that GWO outperforms the KMA and RSA in four test functions of F8, F9, F10, and F11. In contrast, KMA outperforms the GWO and RSA in two test functions of F12 and F13.

The Wilcoxon rank-sum test (WSRT) is employed as the statistical test to confirm the difference between the results produced by different optimizers. From Table IV, we can observe that GWO is significantly better than KMA and RSA for ten benchmark functions (F1, F2-F11), where all the *p*-value is less than 0.05. Meanwhile, KMA is considerably better than GWO and RSA for F12 and F13 functions, where both *p*-values are less than 0.05. Moreover, in the F10 function, GWO vs. RSA is not significantly better, with a *p*-value greater than 0.05. Finally, RSA is substantially better in the F2 function, where the *p*-value is less than 0.05.

Next, all optimization algorithms are compared to each other. For fairness, we create an initial seed population to get good comparison results. Hence, the optimizers used the same γ and C values. The optimizer algorithms and the five datasets were executed for 30 runs. Table V shows the results of the runs. From the table, we can observe that KMA yielded the best mean value for DS3, DS4, and DS5, while GWO yielded the best mean value for DS1 and DS2. Nevertheless, based on the statistical test (see Table VI), only KMA gained significant differences from RSA and GWO for DS4 and DS5. These results revealed that separating the dataset based on their project size improved performance.

TABLE II. Results of unimodal benchmark functions

Function		KMA	GWO	RSA
F1	Avg	13265.59	**3.19E-48**	67761
	Std	12401.28	1.72E-47	6690.35
F2	Avg	-5.42E+21	-8.89E+20	**-7.20E+22**
	Std	1.76E+22	1.31E+05	1.92E+23
F3	Avg	200564	**1.68E-53**	994189.88
	Std	203888	8.30E-53	118220.83
F4	Avg	37.28	**3.95E-30**	87.37
	Std	29.45	2.05E-29	2.85
F5	Avg	2.5E+07	**28.88**	2.6E+08
	Std	5E+07	0.077	4.7E+07
F6	Avg	12715	**4.47**	67809.64
	Std	13678.36	1.87	6731.36
F7	Avg	27.19	**8.71**	137.75
	Std	20.16	0.41	19.22

TABLE III. Results of multimodal benchmark functions

Function		KMA	GWO	RSA
F8	Avg	-2286.4	**-2686.65**	-2270.11
	Std	411.5	9.09E-13	427.75
F9	Avg	62734.4	**0**	440.17
	Std	4512.9	0	32.25
F10	Avg	19.2	**-1.26**	3.01
	Std	0.08	0.69	8.97
F11	Avg	9.54	**-6.5E+07**	628.16
	Std	29.91	1.96E+08	59.76
F12	Avg	**-4.01E+10**	-3.91E+10	-3.5E+10
	Std	7.82E+09	0	8.5E+09
F13	Avg	**-7.5E+10**	-6.99E+10	-6.1E+10
	Std	1.6E+10	3.052E-05	1.6E+10

TABLE IV. The *p*-values of Wilcoxon rank-sum statistical test for 13 mathematical benchmark functions

Function	KMA vs GWO	KMA vs RSA	GWO vs RSA
F1	2.0E-06	2.0E-06	2.0E-06
F2	**3.7E-01**	2.7E-05	1.2E-05
F3	2.0E-06	2.0E-06	2.0E-06
F4	2.0E-06	4.0E-06	2.0E-06
F5	2.0E-06	2.0E-06	2.0E-06
F6	2.0E-06	2.0E-06	2.0E-06
F7	2.0E-06	2.0E-06	2.0E-06
F8	3.0E-06	1.2E-05	1.1E-04
F9	2.0E-06	2.0E-06	2.0E-06
F10	2.0E-06	1.0E-05	**8.8E-01**
F11	1.0E-05	2.0E-06	2.0E-06
F12	3.7E-05	1.5E-04	2.0E-02
F13	3.1E-05	6.7E-05	6.7E-03

TABLE V. BENCHMARK RESULTS OF OPTIMIZER ALGORITHMS FOR ALL DATASETS

Dataset	Metric	KMA	GWO	RSA
DS1	Best	491.53	502.68	523.18
	Worst	1031.75	956.10	1003.96
	Mean	675.21	**672.61**	689.53
	Stdev	108.62	114.64	139.95
DS2	Best	382.85	394.37	379.29
	Worst	952.13	769.06	805.71
	Mean	520.18	**507.49**	529.41
	Stdev	128.74	82.31	105.02
DS3	Best	435.22	438.61	443.50
	Worst	906.64	940.33	951.48
	Mean	**590.56**	605.32	595.39
	Stdev	144.94	117.27	118.50
DS4	Best	262.29	263.95	316.59
	Worst	817.22	588.75	450.83
	Mean	**365.30**	366.47	387.97
	Stdev	121.11	79.04	33.85
DS5	Best	571.09	593.20	644.29
	Worst	1528.85	1205.15	935.98
	Mean	**739.88**	768.43	786.17
	Stdev	188.50	159.17	77.312
	FMR	**1.82**	**1.94**	**2.24**
	Rank	**1**	**2**	**3**

TABLE VI. THE *P*-VALUES OF WILCOXON RANK-SUM STATISTICAL TEST OF KMA, GWO, AND RSA USING FIVE DATASETS

Dataset	KMA vs. GWO	KMA vs. RSA	GWO vs. RSA
DS1	9.3E-01	6.9E-01	6.1E-01
DS2	6.3E-01	6.0E-01	1.6E-01
DS3	6.9E-01	8.3E-01	9.8E-01
DS4	3.8E-01	**3.0E-02**	**4.7E-02**
DS5	1.8E-01	**1.1E-02**	3.8E-01

Besides the performance results, we further explore the diversity analysis of the best solution for the benchmark algorithms for DS4 and DS5. Fig. 1 shows that the median of KMA is smaller compared with GWO and RSA. Based on the interquartile box, we can observe that KMA has a smaller shape than GWO, indicating that KMA has a lower spread.

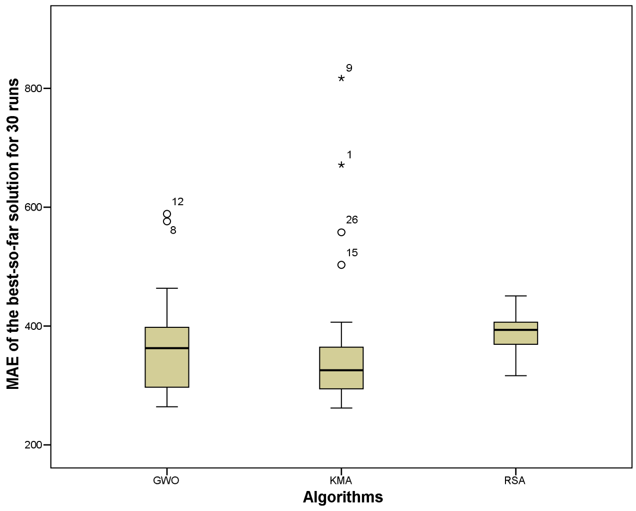

Fig. 1. Best solution diversity analysis for DS4

Finally, Fig. 2 describes the diversity analysis of DS5. Based on the boxplot of the figure, we observed that KMA has the lowest median compared with GWO and RSA. Based on the interquartile box, we can find that KMA has a smaller shape compared with GWO. This shape indicated that KMA has a lower spread and reliable results.

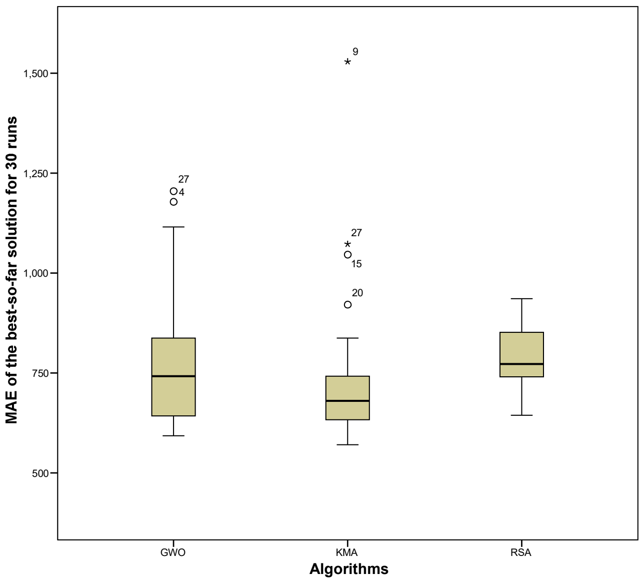

Fig. 2. Best solution diversity analysis for DS5

V. CONCLUSION

This study shows that Komodo *Mlipir* Algorithm (KMA) performed significantly better than Reptile Search Algorithm (RSA) for DS4 and DS5 as of 0.03 ($p < 0.05$). Meanwhile, there were no significant differences in accuracy performance between all pairs of optimizer algorithms for DS1, DS2, and DS3 with *p*-values of WSRT greater than 0.05. We can observe that there was no significant difference between KMA and Grey Wolf Optimizer (GWO) for all datasets. Overall, KMA yielded Friedman mean rank of 1.82, with *p*-values of the Wilcoxon rank-sum rank test of less than 0.05.

According to this result, our work highlights the critical role of optimization algorithms in this field. The implication of this study is that a software organization project manager can utilize this hybrid approach to estimate the future project using their historical dataset. Furthermore, classifying the dataset based on the project size leads to better performance. For future work, adding more heterogeneous datasets and other recent optimizer algorithms would be better to achieve the more robust and confident performance of this hybrid approach.

REFERENCES

[1] G. Karner, "Resource Estimation for Objectory Projects." 1993.

[2] G. Schneider and J. P. Winters, *Applying Use Cases: A Practical Guide*, 2nd ed. Addison Wesley, 2001.

[3] Sholiq, T. Sutanto, A. P. Widodo, and W. Kurniawan, "Effort Rate on Use Case Point Method for Effort Estimation of Website Development," *J. Theor. Appl. Inf. Technol.*, vol. 63, no. 1, pp. 209–218, 2014.

[4] A. B. Nassif, "Towards an Early Software Estimation Using Log-Linear Regression and a Multilayer Perceptron Model," *J. Syst. Softw.*, vol. 86, no. 1, pp. 144–160, 2013.

[5] M. Azzeh and A. B. Nassif, "A hybrid model for estimating software project effort from Use Case Points," *Appl. Soft Comput. J.*, vol. 49, pp. 981–989, 2016, doi: 10.1016/j.asoc.2016.05.008.

[6] A. B. Nassif, D. Ho, and L. F. Capretz, "Towards an early software

estimation using log-linear regression and a multilayer perceptron model," *J. Syst. Softw.*, vol. 86, no. 1, pp. 144–160, Jan. 2013, doi: 10.1016/j.jss.2012.07.050.

[7] S. Banitaan, A. B. Nassif, and M. Azzeh, "Class Decomposition Using K-Means and Hierarchical Clustering," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2015, pp. 1263–1267, doi: 10.1109/ICMLA.2015.169.

[8] S.-W. Lin, K.-C. Ying, S.-C. Chen, and Z.-J. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector machines," *Expert Syst. Appl.*, vol. 35, no. 4, pp. 1817–1824, Nov. 2008, doi: 10.1016/j.eswa.2007.08.088.

[9] C.-L. Huang and J.-F. Dun, "A distributed PSO–SVM hybrid system with feature selection and parameter optimization," *Appl. Soft Comput.*, vol. 8, no. 4, pp. 1381–1391, Sep. 2008, doi: 10.1016/j.asoc.2007.10.007.

[10] X. Liu and H. Fu, "PSO-based support vector machine with cuckoo search technique for clinical disease diagnoses," *Sci. World J.*, vol. 2014, 2014, doi: 10.1155/2014/548483.

[11] A. M. Al-Zoubi, M. A. Hassonah, A. A. Heidari, H. Faris, M. Mafarja, and I. Aljarah, "Evolutionary competitive swarm exploring optimal support vector machines and feature weighting," *Soft Comput.*, vol. 25, no. 4, pp. 3335–3352, Feb. 2021, doi: 10.1007/s00500-020-05439-w.

[12] J. Zhou *et al.*, "Optimization of support vector machine through the use of metaheuristic algorithms in forecasting TBM advance rate," *Eng. Appl. Artif. Intell.*, vol. 97, no. September 2020, p. 104015, 2021, doi: 10.1016/j.engappai.2020.104015.

[13] S. Anupam and A. K. Kar, "Phishing website detection using support vector machines and nature-inspired optimization algorithms," *Telecommun. Syst.*, vol. 76, no. 1, pp. 17–32, Jan. 2021, doi: 10.1007/s11235-020-00739-w.

[14] E. H. Houssein, M. E. Hosney, D. Oliva, W. M. Mohamed, and M. Hassaballah, "A novel hybrid Harris hawks optimization and support vector machines for drug design and discovery," *Comput. Chem. Eng.*, vol. 133, p. 106656, Feb. 2020, doi: 10.1016/j.compchemeng.2019.106656.

[15] A. Tharwat and T. Gabel, "Parameters optimization of support vector machines for imbalanced data using social ski driver algorithm," *Neural Comput. Appl.*, vol. 32, no. 11, pp. 6925–6938, Jun. 2020, doi: 10.1007/s00521-019-04159-z.

[16] M. R. Braz and S. R. Vergilio, "Using fuzzy theory for effort estimation of object-oriented software," in *16th IEEE International Conference on Tools with Artificial Intelligence*, 2004, no. Ictai, pp. 196–201, doi: 10.1109/ICTAI.2004.119.

[17] A. Ardiansyah, R. Ferdiana, and A. E. Permanasari, "MUCPSO: A Modified Chaotic Particle Swarm Optimization with Uniform Initialization for Optimizing Software Effort Estimation," *Appl. Sci.*, vol. 12, no. 3, p. 1081, Jan. 2022, doi: 10.3390/app12031081.

[18] G. Robiolo and R. Orosco, "Employing use cases to early estimate effort with simpler metrics," *Innov. Syst. Softw. Eng.*, vol. 4, no. 1, pp. 31–43, Apr. 2008, doi: 10.1007/s11334-007-0043-y.

[19] P. Mohagheghi, B. Anda, and R. Conradi, "Effort estimation of use cases for incremental large-scale software development," in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 2005, pp. 303–311, doi: 10.1109/ICSE.2005.1553573.

[20] B. Anda, H. Dreiem, D. I. K. Sjoberg, and M. Jorgensen, "Estimating Software Development Effort based on Use Cases – Experiences from Industry," in *The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, C. G. Kobryn, Ed. Springer Berlin Heidelberg, 2001.

[21] B. Anda, E. Angelvik, and K. Ribu, "Improving Estimation Practices by Applying Use Case Models," *Prod. Focus. Softw. Process Improv.*, vol. 2559, no. 1325, pp. 383–397, 2002, doi: 10.1007/3-540-36209-6_32.

[22] M. Ochodek, J. Nawrocki, and K. Kwarciak, "Simplifying effort estimation based on Use Case Points," *Inf. Softw. Technol.*, vol. 53, no. 3, pp. 200–213, Mar. 2011, doi: 10.1016/j.infsof.2010.10.005.

[23] M. Ochodek, B. Alchimowicz, J. Jurkiewicz, and J. Nawrocki, "Improving the reliability of transaction identification in use cases," *Inf. Softw. Technol.*, vol. 53, no. 8, pp. 885–897, 2011, doi: 10.1016/j.infsof.2011.02.004.

[24] H. L. T. K. Nhung, V. Van Hai, R. Silhavy, Z. Prokopova, and P. Silhavy, "Parametric Software Effort Estimation Based on Optimizing Correction Factors and Multiple Linear Regression," *IEEE Access*, vol. 10, pp. 2963–2986, 2022, doi: 10.1109/ACCESS.2021.3139183.

[25] A. B. Nassif, L. F. Capretz, D. Ho, and M. Azzeh, "A Treeboost Model for Software Effort Estimation Based on Use Case Points," in *2012 11th International Conference on Machine Learning and Applications*, Dec. 2012, no. December 2012, pp. 314–319, doi: 10.1109/ICMLA.2012.155.

[26] I. Jacobson, "Object Oriented Development in an Industrial," 1987.

[27] I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley, 1992.

[28] A. B. Nassif, "Software Size and Effort Estimation from Use Case Diagrams Using Regression and Soft Computing Models," The University of Western Ontario, 2012.

[29] S. S. Keerthi and C. Lin, "Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel," *Neural Comput.*, vol. 15, no. 7, pp. 1667–1689, 2003, doi: 10.1162/089976603321891855.

[30] J. Zhou *et al.*, "Optimization of support vector machine through the use of metaheuristic algorithms in forecasting TBM advance rate," *Eng. Appl. Artif. Intell.*, vol. 97, Jan. 2021, doi: 10.1016/j.engappai.2020.104015.

[31] S. Suyanto, A. A. Ariyanto, and A. F. Ariyanto, "Komodo Mlipir Algorithm," *Appl. Soft Comput.*, vol. 114, no. xxxx, p. 108043, Jan. 2022, doi: 10.1016/j.asoc.2021.108043.

[32] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014, doi: 10.1016/j.advengsoft.2013.12.007.

[33] L. Abualigah, M. A. Elaziz, P. Sumari, Z. W. Geem, and A. H. Gandomi, "Reptile Search Algorithm (RSA): A nature-inspired meta-heuristic optimizer," *Expert Syst. Appl.*, vol. 191, no. November, p. 116158, Apr. 2022, doi: 10.1016/j.eswa.2021.116158.

[34] Xin Yao, Yong Liu, and Guangming Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, Jul. 1999, doi: 10.1109/4235.771163.

[35] J. G. Digalakis and K. G. Margaritis, "On benchmarking functions for genetic algorithms," *Int. J. Comput. Math.*, vol. 77, no. 4, pp. 481–506, Jan. 2001, doi: 10.1080/00207160108805080.