Feature selection optimisation of software product line using metaheuristic techniques

Hitesh Yadav*

Department of Computer Science and Engineering,
The NorthCap University,
Gurugram, India
Email: hiteshi.yadav@gmail.com
*Corresponding author

A. Charan Kumari

Faculty of Engineering,
Dayalbagh Educational Institute,
Dayalbagh, Agra, India
Email: charankumari@yahoo.co.in

Rita Chhikara

Department of Computer Science and Engineering,
The NorthCap University,
Gurugram, India
Email: ritachhikara@ncuindia.edu

Abstract: The role of software product line (SPL) is very important in representing the same system with multiple variants. Feature models are used to define SPL. In this paper, genetic algorithm (GA), hyper-heuristic algorithm and particle swarm optimisation (PSO) have been applied for feature selection optimisation in SPL. Also, an improved fitness function is applied for optimisation of features in SPL. The objective function is designed by taking reusability and consistency of features (components) into consideration. Furthermore, we have used a case study and discussed about software product line in detail. A non-parametric test, i.e., Kruskal-Wallis test has been performed to analyse performance and computation time of 20 to 1,000 features sets and identify core features. Through extensive experimental analysis, it is observed that PSO outperforms GA and hyper-heuristic algorithm.

Keywords: genetic algorithm; product line; feature model; particle swarm optimisation; PSO; software product line; SPL; hyper-heuristic evolutionary algorithm.

Reference to this paper should be made as follows: Yadav, H., Kumari, A.C. and Chhikara, R. (2020) 'Feature selection optimisation of software product line using metaheuristic techniques', *Int. J. Embedded Systems*, Vol. 13, No. 1, pp.50–64.

Biographical notes: Hitesh Yadav is pursuing her PhD and is an MTech holder in Computer Science. She has 6+ years' experience. She got second position in BE at university level. She was also awarded scholarship in BE. She is a CISCO certified Training Instructor for CCNA module-1, 2, 3 and 4. She has been awarded as an Advanced Level Instructor in 2017. She has published many research papers in international conferences including IEEE, Elsevier. Her areas of interests are on software engineering, computer network and optimisation technique. She has guided many BTech Projects and MTech Thesis.

A. Charan Kumari received her PhD from Dayalbagh Educational Institute, in collaboration with Indian Institute of Technology, Delhi, India, under their MOU. She has an excellent teaching experience of 17 years in various esteemed institutions and received various accolades including the best teacher award. Her current research interests include search based software engineering, evolutionary computation and soft computing techniques. She has published papers in journals and conferences of national and international repute. She has also served as reviewer of various journals and conferences. She has delivered an invited talk at 43rd CREST Open Workshop on Hyper-heuristics at University College London, London. She served as a Technical Committee Chair at International Conference on Computational Intelligence and Data Science (ICCIDS 2018). She is a member of IEEE, Computer Society of India (CSI) and Systems Society of India (SSI).

Rita Chhikara is an Assistant Professor in Department of CSE/IT. She has an excellent teaching experience of more than 20 years in various esteemed institutions. She received her BTech from Pune University, MTech CSE degree from Punjab Technical University, MBA in IT from Sikkim and Manipal University and has completed her PhD from The Northcap University (NCU) in Data Mining. She is presently working on a project titled 'Neural network based steganalysis' funded by Department of Science and Technology. Her current areas of research include data mining, pattern recognition, machine learning, image processing and deep learning. She has guided around 17 BTech projects, 12 MTech Thesis. She has published 28 papers in peer reviewed international journals with good indexing and reputed national/international conference proceedings. She has chaired a session at IEEE conference, ISMS Malaysia 2015. She is member of ISTE, ACM and IEEE.

This paper is a revised and expanded version of a paper entitled 'Selection optimization in SPL using genetic algorithm' presented at Conference on Computational Intelligence and Data Science, Gurgaon, India, 7–8 April 2018.

1 Introduction

The software product line (SPL) is a part of a software-intensive system. The software product family consists of multiple product lines. Product lines share similar types of features. In order to use SPLs in a productive manner, researchers and industrialists are continuously working on SPL enhancements (Henard et al., 2013).

In today's lifetime, as the customer's requirements are constantly evolving so software development cost, software complexity and software delivery time are increased.

SPLs take more time in designing and development stages as compared to the development of individual product line.

Many existing features can be reused in developing software products along with new variant features. Therefore, reusable features of the SPL can be utilised to develop entire new software product(s). The SPL can be used in achieving objectives by reducing development time and cost. Most of the engineers use SPL in organisations to improve productivity and software product quality. (Kumar and Rajkumar, 2017) feature models are used by SPL. Feature model defines parent child relationship among features. The nodes in feature model represent feature and the connection between features defines the relationship. Relationship among features is called constraints. Feature is defined as the characteristics of product. Features selection is dependent on the user's request. Li et al. (2011) presented a technique to transform constraints of feature model into inequality constraints. Following are the ways to represent constraints into inequality constraints.

- Paternity: Assume feature 'm' infers feature 'n', if 'n' is selected, 'm' should be selected together. ($x_m <= x_n$). For instance, if feature *screen* is selected then its parent feature *television* is also chosen together.
- Or: One or more than one node is selected.
- And: Assume feature 'm' infers feature 'n' and 'n' is compulsory node. If 'm' is selected, then 'n' is selected together. ($x_m <= x_n$)
- Imply (require): For node 'm' and 'n', Assume 'm' require 'n'. When 'm' is selected, 'n' must be selected together. For example, a feature *colour* require feature *high definition*.
- Exclude (exclusive): Assume node 'm' and 'n'. 'm' excludes 'n'. Only one feature is chosen. ($x_m + x_n <= 1$). As shown in Figure 1, only one feature is selected from feature *battery* and feature *electricity*.

There are different types of constraints used in Figure 1 such as optional, mandatory, inclusive, exclusive, require and so on.

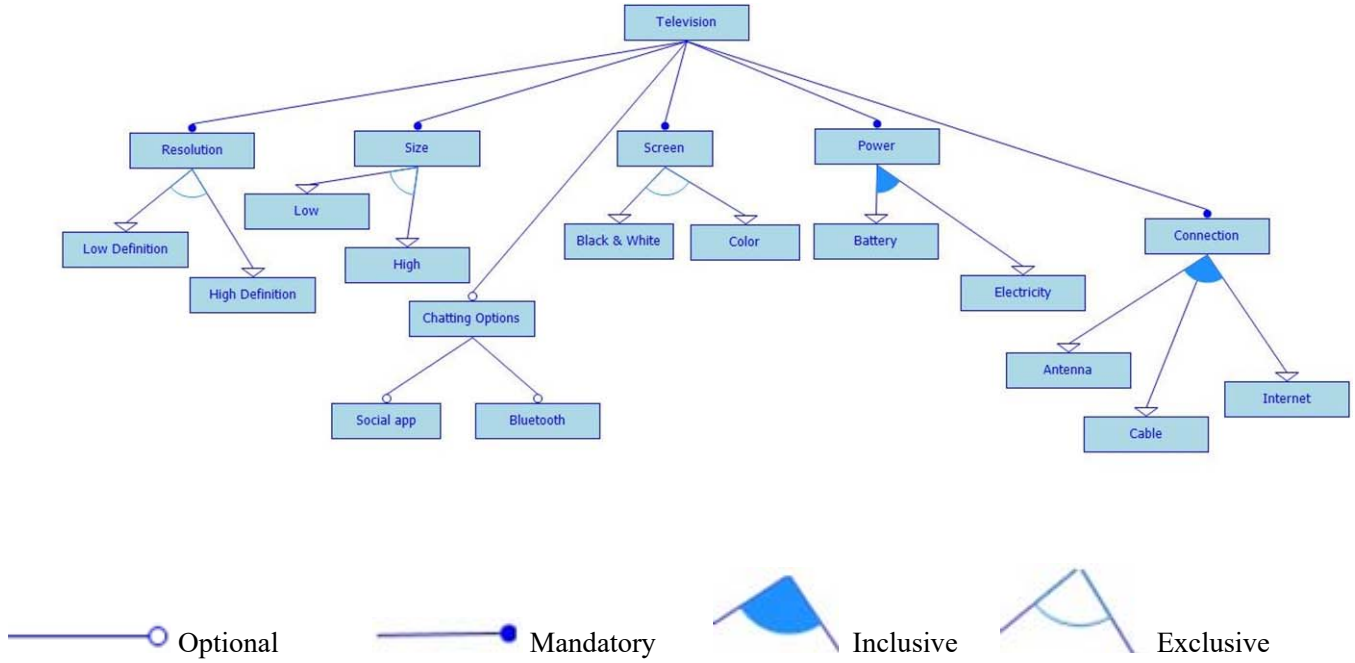
Optional features mean a product that can be developed with or without features. Some of the examples of optional features are *chatting options*, *social app* etc. Mandatory features are those features which must be part of the product. Some of the examples of mandatory features are *resolution*, *size* and *screen*, etc. In exclusive or alternative, single node is selected if the parent node is chosen. Inclusive means at least one node is chosen if parent node is chosen. Require is a part of cross constraint such as *social app* features requires *internet* feature as shown in Figure 1.

A product is defined as a collection of features. + and - signs are used to represent features in product line. + sign indicates that node is selected, and its value is set 1/true for the specific product line and – sign indicates that node should not be selected or its value is 0/false. Following are some of the product lines (P_1, P_2, P_3, P_4 and P_5 …… P_n) which can be created by selecting features from feature model shown in Figure 1.

$$P_1 = \begin{Bmatrix} +Television + Resolution - Low\ Definition \\ +High\ Definition + Size + High - Chatting\ Options \\ +Screen + Colour + Power + Battery + Connection \end{Bmatrix}$$

$$P_2 = \begin{Bmatrix} +Television + Resolution + High\ Definition \\ -Low\ Definition + Size - Low + High - Chatting\ Options \\ +Social\ App + Screen + Colour + Power \\ +Battery + Connection \end{Bmatrix}$$

Figure 1 Feature model of television (see online version for colours)



$$P_3 = \begin{Bmatrix} +Television + Resolution - Low\ Definition \\ +High\ Definition + Size + High + Chatting\ Options \\ +Bluetooth + Screen + Black\ and\ White + Power \\ +Electricity + Battery + Connection + Cable \end{Bmatrix}$$

$$P_4 = \begin{Bmatrix} +Television + Resolution - Low\ Definition \\ +High\ Definition + Size + High + Chatting\ Options \\ +Bluetooth + Screen + Black\ and\ White + Power \\ +Electricity + Battery + Connection + Internet \end{Bmatrix}$$

$$P_5 = \begin{Bmatrix} +Television + Resolution - Low\ Definition \\ -High\ Definition + Size + High + Chatting\ Options \\ +Bluetooth + Screen + Black\ and\ White + Power \\ +Electricity + Battery - Connection + Bluetooth \end{Bmatrix}$$

$$P_n = \begin{Bmatrix} +Television + Resolution - Low\ Definition \\ +High\ Definition + Size + High + Chatting\ Options \\ +Bluetooth + Screen + Black\ and\ White + Power \\ +Electricity + Connection + Internet \end{Bmatrix}$$

Resolution, *size*, *screen*, *power* and *connection* are the mandatory nodes so these nodes must be part of each product lines. Product line P_1 indicates *television*, *resolution*, *high definition*, *size*, *high*, *screen*, *colour*, *battery*, *power*, *connection* features are assigned 1 or selected for P_1 product line, while the other features such as *low definition*, *chatting options* are assigned 0 or not selected for the respective product line. Product line P_4 indicates that if the feature *social app* is selected then feature internet must be selected because feature *social app* cannot be used with feature *internet*.

Few features are enough to produce a product. With 'x' set of features, 2^x products can be developed. Therefore, hundreds or thousands of software products can be generated using a few features. By combining a small number of features an exponential number of products can be produced. Since it is a NP-hard problem, therefore an optimisation technique is used to find an optimal solution (Guo et al., 2011). It improves time complexity and reduces cost and effort. Most of the researchers used genetic algorithm with multiple sets of the feature models. We have applied genetic algorithm, PSO and hyper-heuristic to compute time and effort. Kruskal-Wallis test is a non-parametric test which is applied on multiple feature sets.

Guo et al. (2011) focused on genetic algorithm for feature selection optimisation in SPL. It is a very complicated task to optimise feature with resource constraints. Also, these methods have exponential complexity and have not considered extended size feature sets. Authors have considered case study based on database with CPU and cost constraints. Authors used large dataset of feature model for optimisation. White et al. (2009) applied a polynomial time method on large feature sets which were incorporated with resource constraints. Addition of resource constraints made optimisation more difficult. Many researchers have worked with methods using SAT solver (Batory, 2005) BDD (Czarnecki and Wasowski, 2007), CSP (Benavides et al., 2005) without considering resource constraints.

Guo et al. (2011) defined automatic product derivation using genetic algorithms for feature selection optimisation problem in SPL. GAFES improves up to 99% by shortening the time than the traditional algorithm. Alidra and Kimour (2014) decided reusing of software products in the system.

At design level, the author explained the concept of reusable modules. Authors determined a family of genetic algorithm and obtained a line of genetic algorithms from that family.

Afzal et al. (2014) defined the real FM, and optimised a group of feature models consisting of a small set of feature models (about 100 features) and two large sets of 500 to 1,000 feature models. Authors used genetic algorithms to minimise inconsistency in features such as substitution, mandatory, and comprehensive constraints. Wang and Pang (2014) compared two different technologies. One is genetic algorithm and the other is a filtered Cartesian smoothing (FCF). They used genetic algorithm for feature selection. Experimental results proved that there is a ten percent improvement in results as compare to genetic algorithm with more run time. Chohan et al. (2017) applied a multi-objective optimisation method and proposed a framework for improving SPL. Sharma et al. (2014) defined business process modelling, which has used SPL s in defining notation. They introduced different dimensions and expanded to a configurable format generated configurable business process modelling notation (C-BPMN) and classified all feasible variants of an enterprise system. Authors have generated the tool and utilised it to determine the SPL. Sharma et al. (2015) proposed C-BPMN and individualised the configurable processes for each requirement. This paper compared the C-BPMN with configurable event process chain.

Abbas et al. (2016) focused on cross-cutting concern. Authors found that finding and modularise these concerns at modelling level is important. Authors applied Union-find algorithm for finding cross-cutting concerns in Feature model. Further, author used genetic algorithm for optimisation of features selection. Saka and Dogan (2012) explained survey on developments in metaheuristic algorithms. Authors explained new swarm intelligence algorithms, i.e., cuckoo search algorithms, firefly algorithms, perform better than genetic algorithm. Authors provided survey of ten recent metaheuristic algorithms.

Wu et al. (2008) worked on the problem of grid technologies. Authors has focused on to design of the genetic algorithm based scheduling strategies by taking care of four different fault tolerance methods which are retry, replication, migration and checkpoint.

Linsbauer et al. (2014) analysed seventeen feature models using genetic programming and compared results with previous results. Mamun et al. (2016) analysed next release problem using multi-objective genetic algorithms. The author also modified mathematical model and measure feature objectives for obtaining sub-optimal results. Ferreira et al. (2017) applied four multi objective algorithms by utilising three and four objectives. Filho et al. (2018) introduced a user preference based hyper-heuristic technique. The authors hybridised non-dominated sorting genetic algorithm (NSGA)-II with hyper-heuristic and compared with NSGA-II and proved that r-NSGA-II-HH outperform traditional approach. Sayyad et al. (2013) identified five objectives and applied multi-objective optimisation algorithms. Sahin (2014) explained various feature selection methods such as filter methods, wrapper methods, heuristic search algorithms and embedded methods. Authors applied feature selection techniques on datasets and provided explanations of applicability of these techniques.

Metaheuristic algorithm, i.e., particle swarm optimisation (PSO) and GA were applied by Tsafarakis et al. (2011) for optimal product line design and compared algorithms using Monte Carlo simulation method. Authors proved that PSO performance was better than GA. Authors approach is explained by using a real-world case study.

Abdelhalim et al., (2006) applied GA and PSO on large hardware software partitioning problem and find out PSO outperformed GA in the terms of cost and execution time. Also, author carried out several tests using hybrid algorithms. Saed and Kadir (2011) applied GA and PSO to optimise software performance prediction. Authors concluded that after applying these techniques on multiple problems, PSO technique performed better than genetic algorithm.

To find out the highly used product line, it is important to measure each feature. Therefore, certain parameters are required to quantify each feature. Ding et al. (2011) defined the measurement parameters of core asset, i.e., reusability and consistency. These parameters can be used in measurement of contribution of the feature. By computing and adding the contribution of all feature used in SPL, one can get the total contribution of SPL. In this paper, we have used the parameter 'reusability' and 'consistency' to define an improved fitness function. This fitness function has been used by three metaheuristic algorithms namely genetic algorithm, PSO, and Hyper- heuristic. As per literature available, reusability and consistency have not been applied on SPL using fitness function.

The paper is designed section wise. Section 2 elaborates methodology. Section 3 presents results and analysis of genetic algorithm and PSO which is used to solve this problem in the SPL. Lastly, Section 4 briefs conclusion and future work.

2 Methodology

In this study, three algorithms have been applied

a genetic algorithm

b particle swarm algorithm

c hyper-heuristic.

These algorithms have been used to compute performance and computation time by using Kruskal-Wallis test. The next three subsections provide brief introduction of genetic algorithm, PSO, and hyper-heuristic respectively. All techniques use same objective function as defined in Subsection 2.4

2.1 Genetic algorithm

Holland (1975) developed GA by Darwinian evolution. Genetic algorithms have probabilistic search-based procedures to define ways to resolve search-based problems (Alidra and Kimour, 2014). According to researchers' genetic algorithms are applicable for traversing the search space and come together with good results against difficult and nonlinear issues (Alidra and Kimour, 2014). Metaheuristic techniques are used to solve nonlinear problems. GA is the type of metaheuristic algorithm (Alidra and Kimour, 2014). The GA initialises a population called as chromosome (Filho et al., 2018; Guo et al., 2017). The binary value (0, 1) is used to define chromosome (Kumari, 2018).

The purpose is to define an initial subset of populations for optimisation. After every generation a new chromosome (child) is generated from the old chromosome (parent). After computing the fitness function, it becomes the parent offspring (Alidra and Kimour, 2014; Sushama and Reddy, 2018).

There are multiple ways to choose chromosomes, i.e., tournament selection method, elite selection, and roulette wheel selection but we used a roulette wheel selection. The crossover divides the chromosomes, exchanges them, and tries to generate descendants with the highest fitness value. According to a fixed predetermined mutation rate, the mutation chooses the single genes and arbitrarily changes a part of the encoded solution (Alidra and Kimour, 2014). According to the exchange policy, the newly generated offspring replace the individuals before the next generation.

This evolution continues until the chromosome meets certain criteria or until the last generation is finished. The algorithm 1 is defined (Guo et al., 2011) in following way.

Algorithm 1
Step 1 Initialise size of population, generation, crossover probability, and mutation probability.
Step 2 Initialise an individual randomly.
Step 3 Compute fitness value.
Step 4 Select chromosome from population for operation using roulette wheel selection.
Step 5 Generate offspring by executing crossover and mutation probability.
Step 6 Generate new chromosome from old chromosomes.
Step 7 Go to step 3 if the criteria are satisfied.
Step 8 Till the best chromosome is found.

2.2 Particle swarm optimisation

PSO is a metaheuristic algorithm designed by Eberhart and Kennedy (1995). Many applications of PSO have been explained by researchers in continuous domains, but in recent years some work has been published on discrete domains. PSO has wide use therefore researchers have done lots of comparison with other metaheuristic algorithm (Tsafarakis et al., 2011). The Discrete PSO has been found to be successfully applied in the field of feature selection, because it defines an optimal set of features as a candidate solution with good performance (Chhikara et al., 2016; Biswas et. al., 2018). Every single candidate solution determines an individual swarm of flocks that act as particles in the search space (Cheng et al., 2018). Each particle refers a fitness value calculated with the help of the fitness function which is to be optimised (Tu et al., 2006) Velocity indicates the movement of the particle. According to own experience and neighbourhood experiences, each particle adjusted its position and encountered the best position (Eberhart and Kennedy, 1995).

First, a population of particles is randomly generated on the search space. At every iteration, each particle defines two updated 'best' values called global best *gbest* and personal best *pbest*. The binary value (0, 1) is used to define the position of the particles in the discrete PSO. The position of the nth particle of the swarm is (Eberhart and Kennedy, 1995)

$$pos_n(k) = (ps_{n1}, ps_{n2}, \cdots\cdots\cdots\cdots ps_{nd}) \tag{1}$$

and velocity (Eberhart and Kennedy, 1995) is defined as

$$v_n(k) - (v_{n1}, v_{n2}, \cdots\cdots\cdots v_{nd}) \tag{2}$$

And value of iteration is k, for population size p, where $n = 1 \ldots\ldots\ldots p$.

Sigmoid function s_i is used to convert velocity from real number into discrete and update the position of the particle. A sigmoid function is defined as follows. (Tsafarakis et al., 2011)

$$s_i(k) = \frac{1}{(1 + e^{-v_n(k)})} \tag{3}$$

$$pos_n(k+1) = \begin{Bmatrix} 1 & if\ r_n(k) < (S_i(k+1)) \\ 0 & \end{Bmatrix} \tag{4}$$

where r_n is a uniform random number.

Two components are defined to update the value of velocity vector, first is cognitive and other is social. Cognitive defines the distance of the i^{th} particle from its personal best value called p-best. (Eberhart and Kennedy, 1995).

$$pbest_i(k) = (pb_{i1}, pb_{i2}, \cdots\cdots\cdots ps_{id}) \tag{5}$$

and social component define the distance from its global best value called *gbest*.

At $k + 1$ iteration value of *gbest* PSO velocity is updated with the following formula.

$$\begin{aligned} v_i(k+1) = w * v_i(k) + c_1 r_1(k)[pbest_i(k) - pos_i(k)] \\ + c_2 r_2(k)[gbes_i(k) - pos_i(k)] \end{aligned} \tag{6}$$

$$pbest_i(k+1) = \begin{Bmatrix} pos_i(k+1) & if\ fit(pos_i(k+1)) \\ pbest_i(k) & if\ fit(pos_i(k+1)) \end{Bmatrix} \tag{7}$$

fit refers fitness function.

Gbest(k) defines global best position which is defined at iteration k.

$$Gbest(k) = \max\left(fit_1(k)\, fit_2(k) \cdots\cdots fit_m(k)\right) \quad (8)$$

Gbest is defined by best position of the particle. The objective function defined for our problem is given in subsequent sections. The algorithm 2 is defined (Engelbrecht, 2007) in following way.

Algorithm 2
Step 1 Initialise population, velocity, and positions
Step 2 for each particle (number of generations)
Step 3 calculate fitness function.
Step 4 evaluate position.
Step 5 repeat
Step 6 for (each particle)
{
if {pbest_n>pbest}
then
pbest=pbest_n
else
consider previous pbest
Step 7 assign pbest value to gbest
Step 8 end
}
Step 9 For (each particle)
{
update velocity of each particle using equation (6).
update position using equation (4).
}
Step 10 end
Step 11 until (criteria is satisfied)
Step 12 end

2.3 Hyper-heuristic evolutionary algorithm

Cowling et al. (2001) developed the term hyper-heuristic. Hyper-heuristic refers 'heuristic to choose heuristic'. A hyper-heuristic decreases the effort needed to discover a solution. Metaheuristic algorithms work over heuristic search space and try to find near optimal solution hyper-heuristic works over heuristic search space and can be used to find a solution to given problem in the search space of heuristic. The hyper-heuristic handles the option and choose which lower level heuristic to be chosen. The binary value (0, 1) is used to define chromosome. A group of low-level heuristics is defined in (Kumari et al., 2013), i.e., EA/Selection _type/Type of Operation. Two types of operations are used in the algorithm, i.e., crossover and mutation. The basic representation can be defined as

- EA/Selection_type/Operation_type1 (Crossover_type)/Operation_type2(mutation_type).

Selection method is used to select the parent population such as rand-to-best and rand. In the rand-to-best method, one of the parent chromosome selected arbitrarily from the population and another parent chromosome is selected based on elite (best one). In rand both parent chromosomes from the population are arbitrarily selected. To generate offspring, different types of crossover types have been applied such as uniform crossover, hybrid crossover 1, hybrid crossover 2.

In uniform crossover, randomly choose each gene from either of the parents which produces two children. The other type of crossover is a hybrid crossover 1 (hc1). In this method hybridise the single-point crossover with uniform crossover. Till crossover point, one of the children is selected by selecting the genes from the one of the parents and other child is selected by choosing the genes from second parent. The remaining genes of both children are chosen from either of parents randomly. The third type of crossover is hybrid crossover 2(hc2), which is defined by hybridise the two-point crossover with uniform crossover. (Kumari and Srinivas, 2016). Till the first crossover point, first child is selected by selecting genes from first parent and second child is selecting by selected the genes from selected parent. Till second crossover point, randomly selected genes of the two children from either of parents and left one gene are selected from their respective parents to the respective child. The algorithm 3 is defined (Kumari et al., 2013) in following way.

Algorithm 3
Step 1 Set size of population, crossover probability, mutation probability.
Step 2 Initialise parent population randomly.
Step 3 Compute the fitness function.
Step 4 Repeat
Step 5 Select one of low-level heuristic
Step 6 Apply the selected one of the low-level heuristics on parent population and generate offspring population
Step 7 Evaluate fitness value of offspring population.
Step 8 Combine parent and offspring population.
Step 9 Execute non-dominated sorting.
Step 10 End repeat.

2.4 Objective function

Assume a feature model which is defined with 'k' features $F = \{f_i\}$ and $1 <= i <= k$. The main aim is to obtain highly reusable and consistent subset of features among all features (Li et al., 2011)

$$F = \max \sum_{i=1}^{k} x_i c_i \quad (9)$$

If the value of x_i is 0, then the feature is not chosen. If x_i is 1 then selected. c_i represents contribution of i^{th} feature. Contribution is calculated with the help of reusability and consistency. The contribution of m^{th} feature is calculated as follows

$$contribution_m = reusalbility_m + consistency_m \quad (10)$$

The improved fitness function is as follows.

$$F = \max \sum_{i=1}^{k} x_i \left(reusability_i + consistency_i \right) \tag{11}$$

2.4.1 Reusability

Reusability (Ding et al., 2011) is the frequency of usage. Frequency of a particular feature could be computed by using equations 12 and 13. Let $U = (u_{ij})$ $n \times m$ be a reusable matrix. Here, 'n' represents a feature, and 'm' represents a product line.

If product line P_j used feature r_i, $u_{ij} = 1$ is considered, otherwise 0 is considered.

$$u_i \sum_{j=1}^{m} u_{ij} \tag{12}$$

A specific feature has reusability value which is specified by equation (13). R_i represents reusability.

$$R_i = \frac{\sum_{j=1}^{m} u_{ij}}{m} \tag{13}$$

In Ding et al. (2011) they have classified features in terms of high to low reusability level (Core Type I, II, III, and IV) (Ding et al., 2011; Yadav and Kumari, 2018). If the reusability value is greater than 0.5 it comes in high reusable category also called as core type I. If the reusability of feature lies in 0.2 to 0.5 then it is called as core type feature II. If the reusability of feature lies in 0.05 to 0.2 then it is called as core type feature III and if the value is less than 0.05 then it called as core type feature IV. Yadav et al. (2018) have considered a case study and identified features which are high reusable and least reusable. Authors have done in-depth analysis of feature model.

2.4.2 Consistency

Consistency, Ding et al. (2011) defined how core features are consistent with the product line. It is calculated by using equation (14).

$$Consistency\left(r_i, r_j \cdots\cdots r_n\right) = \frac{\sum_{n=1}^{m} u_{in} u_{jn} \cdots u_{kn}}{m} \tag{14}$$

3 Experimental results and analysis

This section discusses various parameters used for GA, PSO, and hyper-heuristic. This is followed by comparisons of results of GA, PSO, and hyper-heuristic in terms of mean, standard deviation, and significance level (p-value). Analysis of fitness value with increasing iteration has been depicted through figures for the algorithms in subsequent subsections. All the experiments were implemented in MATLAB.

3.1 Parameters of GA, PSO and hyper-heuristic evolutionary algorithm

The parameter used for PSO, genetic algorithm and hyper-heuristic work well that are defined in Tables 1, 2 and 3 respectively. After thorough experiment it has been observed that the crossover and mutation probability have given better results with high rate convergence. Shi and Eberhart (1988) defined the usage of inertia weight. Inertia weight helps in controlling the impact of previous histories of velocities on current velocity. Inertia weight is defined by a parameter 'w' (Tsafarakis et. al., 2011). In PSO the inertia weight is 1. The size of the population used in PSO, hyper-heuristic, and genetic algorithm is set to 150. A roulette wheel selection method has been used by GA and hyper-heuristic.

Table 1 Parameters used in particle swarm optimisation

Parameters	*Values*
w(inertia weight)	1
c1, c2(acceleration coefficients)	2
Max iteration	20
Population size	150

Table 2 Parameters used in genetic algorithm

Name of parameters	*Data size/method*
Size of population	150
Number of generations	100
Cross-over type	one-point crossover
Parent selection	Roulette-wheel selection
Cross-over probability	0.8
Mutation probability	1.0
Mutation type	Random mutation

Table 3 Parameters used in hyper-heuristic algorithm

Parameters	*Data size/method name*
Size of population	150
Generation size	100
Types of parent selection method	rand, rand-to-best
No of low-level heuristic	12
Types of crossover	Uniform, hybrid crossover type1, hybrid crossover type2.
Mutation	Copy and exchange
Selection method used to select low level heuristic	Roulette wheel selection

Table 4 Comparisons of fitness value obtained by GA, PSO and hyper-heuristic

Number of features	*GA*		*PSO*		*Hyper-heuristic*		*p-value*
	Mean	*Standard deviation*	*Mean*	*Standard deviation*	*Mean*	*Standard deviation*	
20	6.7600000	0.000000	10.130000	0.000000	6.7600000	0.0000000	4.7195e-20
30	11.580000	0.000000	15.180000	0.000000	11.580000	0.0000000	4.7195e-20
50	21.700000	0.000000	25.250000	0.000000	21.700000	0.0000000	4.7195e-20
100	46.380000	0.000000	49.950000	0.000000	46.380000	0.0000000	4.7195e-20
150	71.313331	0.000000	74.651776	0.081550	71.313331	0.0000000	5.9859e-20
200	95.766667	0.000000	99.388222	0.174494	95.766667	0.0000000	1.5237e-19
250	122.27711	0.208536	125.34000	0.306717	122.353329	0.0000000	8.1038e-18
300	145.78399	0.518281	149.45443	0.368283	146.819990	0.0000000	2.4005e-18
350	168.26867	0.784818	175.15045	0.44618	172.383781	0.8046510	5.4071e-18
400	186.86822	1.306626	197.82733	0.462919	191.852662	3.0309860	1.9385e-16
450	206.50933	1.279557	222.44955	0.552921	213.929775	4.0462320	3.8496e-17
500	220.07867	2.004620	247.55433	0.648987	228.928000	4.6449450	2.2612e-17
1,000	396.24844	3.542473	495.54467	1.110090	397.803331	10.163833	6.8206e-14

Figure 2 Boxplot of fitness value of GA, PSO, and hyper-heuristic, (a) 20 features (b) 30 features (c) 50 features (d) 100 features (e) 150 features (f) 200 features (g) 250 features (h) 300 feature (i) 350 feature (j) 400 features (k) 450 features (l) 500 features (m) 1,000 features (see online version for colours)

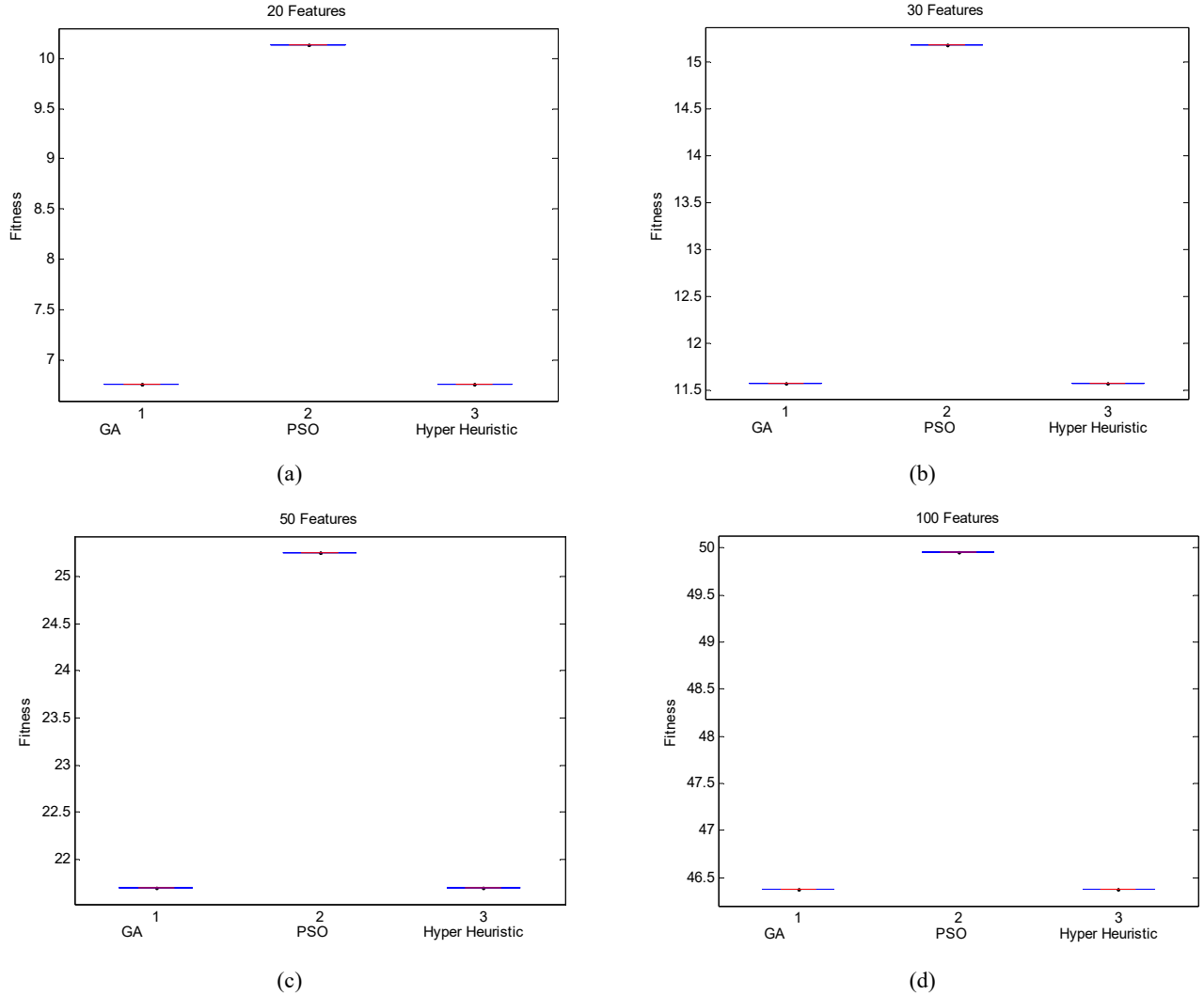
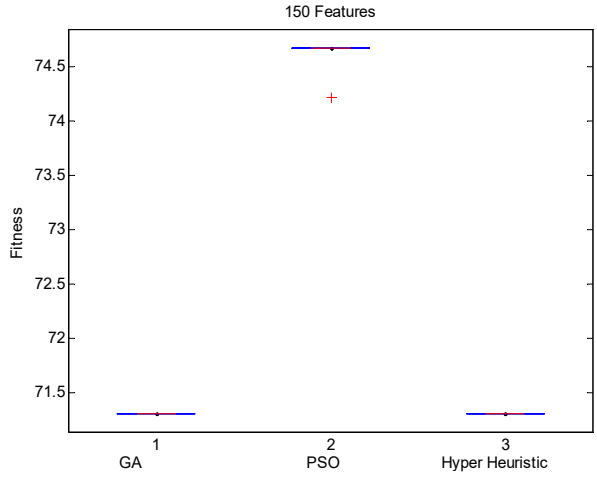
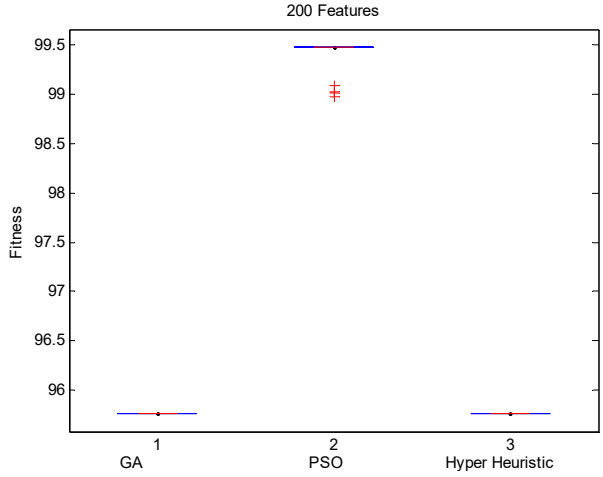


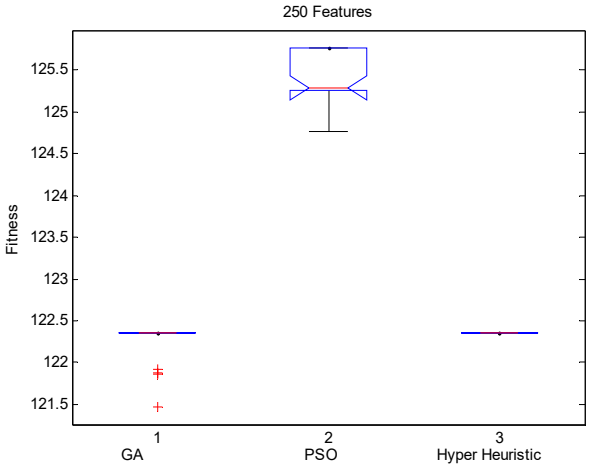
Figure 2 Boxplot of fitness value of GA, PSO, and hyper-heuristic, (a) 20 features (b) 30 features (c) 50 features (d) 100 features (e) 150 features (f) 200 features (g) 250 features (h) 300 feature (i) 350 feature (j) 400 features (k) 450 features (l) 500 features (m) 1,000 features (continued) (see online version for colours)



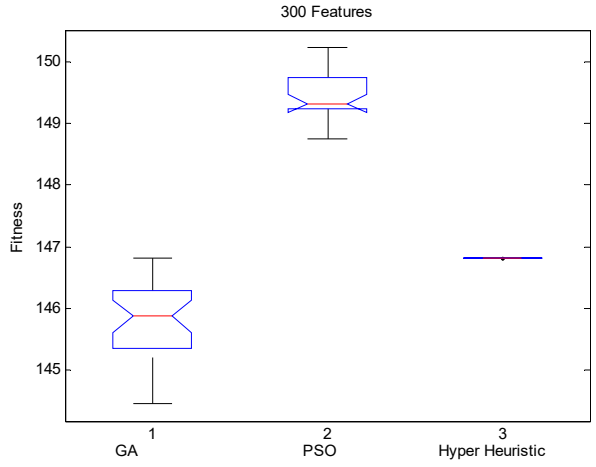
(e)



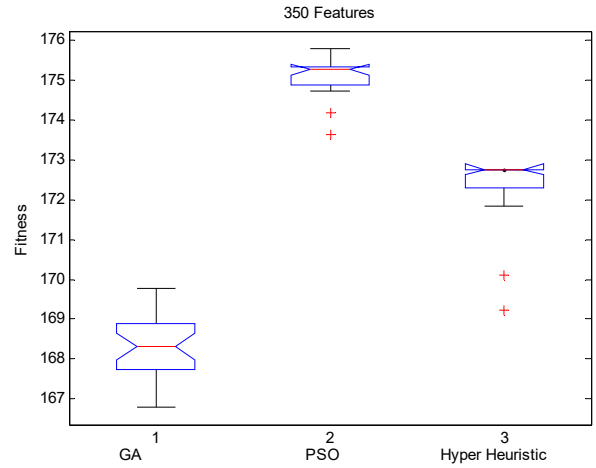
(f)



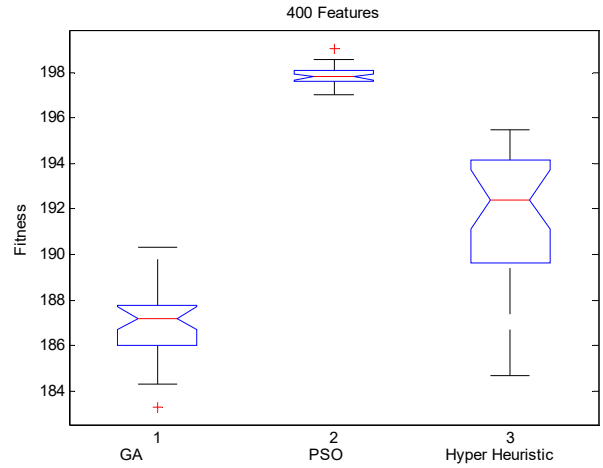
(g)



(h)

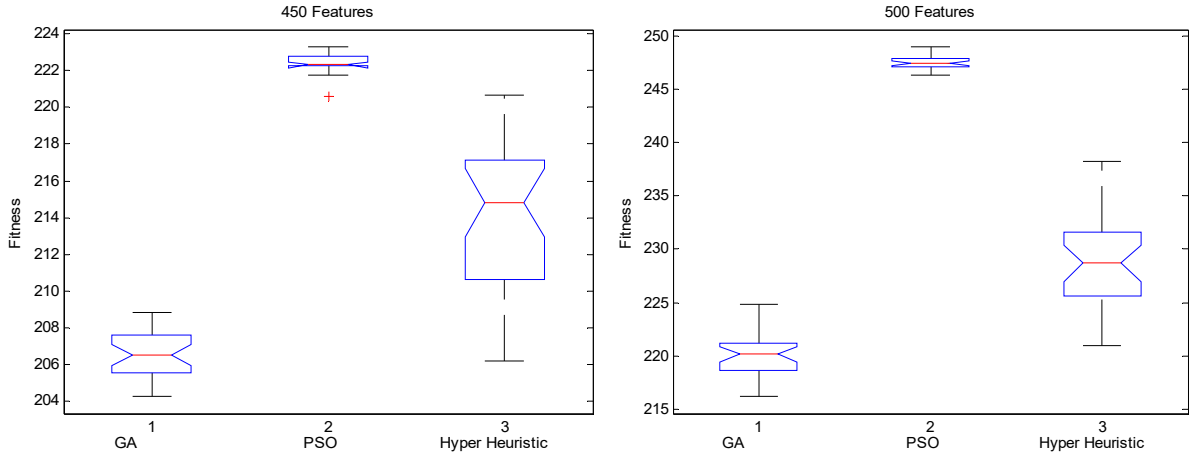


(i)



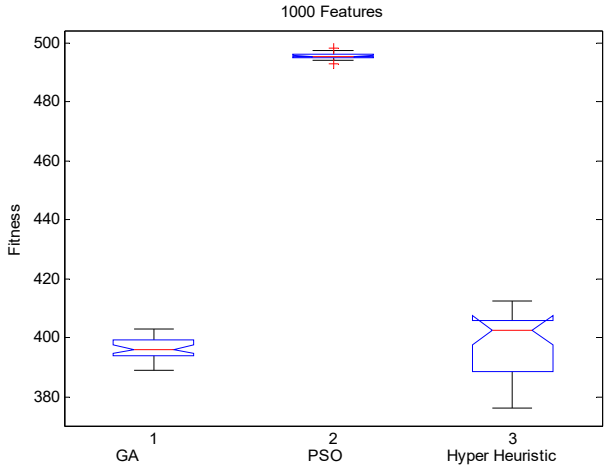
(j)

Figure 2 Boxplot of fitness value of GA, PSO, and hyper-heuristic, (a) 20 features (b) 30 features (c) 50 features (d) 100 features (e) 150 features (f) 200 features (g) 250 features (h) 300 feature (i) 350 feature (j) 400 features (k) 450 features (l) 500 features (m) 1,000 features (continued) (see online version for colours)



(k)

(l)



(m)

3.2 Comparison of GA, PSO and hyper-heuristic

Since all the algorithms are stochastic in nature, therefore each algorithm has been run 30 times on each feature sets to calculate mean and standard deviation. All the feature sets are analysed using non-parametric test, i.e., Kruskal-Wallis test in which it does not need any assumption on the parametric distribution of the samples. This non-parametric test compares the median of the set and provides p-value for null hypothesis. The confidence level has been considered 95% it means p-value should be below 0.05. Table 4 shows the mean and standard deviation of the fitness value acquired by all algorithms along with p-value. From the Table 4, value of mean fitness value of PSO is higher than other algorithms therefore it is clearly seen that fitness value produced by the PSO is showing improvement over genetic algorithm and hyper-heuristic. The results show that p-value are lesser than 0.05.

As a result, the solutions prove that PSO is able to discover the best fitness value. According to the values of standard deviation consistent performance of PSO is perceptible. Using Kruskal-Wallis test in Table 4, it has been analysed that all reported solutions (p-value) are much lesser than 0.05 which proves that performance level of PSO with regard fitness value are significantly different at 95% confidence level. The value of standard deviation is 0 till 200th feature, 100th feature and 300th features for GA, PSO, and hyper-heuristic respectively.

Figure 2 defines boxplots of fitness value of GA, PSO and hyper-heuristic for 20, 30, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, and 1,000 features. In Figure 2, x axis defines three values: 1 refers GA, 2 refers PSO and 3 refers hyper-heuristic, y axis defines fitness value.

Boxplot depict the number of solutions obtained by the implemented algorithms. As shown in Figure 2(a) the PSO boxplot has higher fitness value than GA and hyper heuristic. Again, in Figure 2(b) PSO has higher fitness value than other two algorithms. All the boxplots clearly show superiority of PSO. Also, all boxplots of genetic algorithm, PSO and hyper-heuristic are not having similar shapes and ranges, which means variance is not equal. Thus, PSO has larger range than genetic algorithm and hyper-heuristic. All

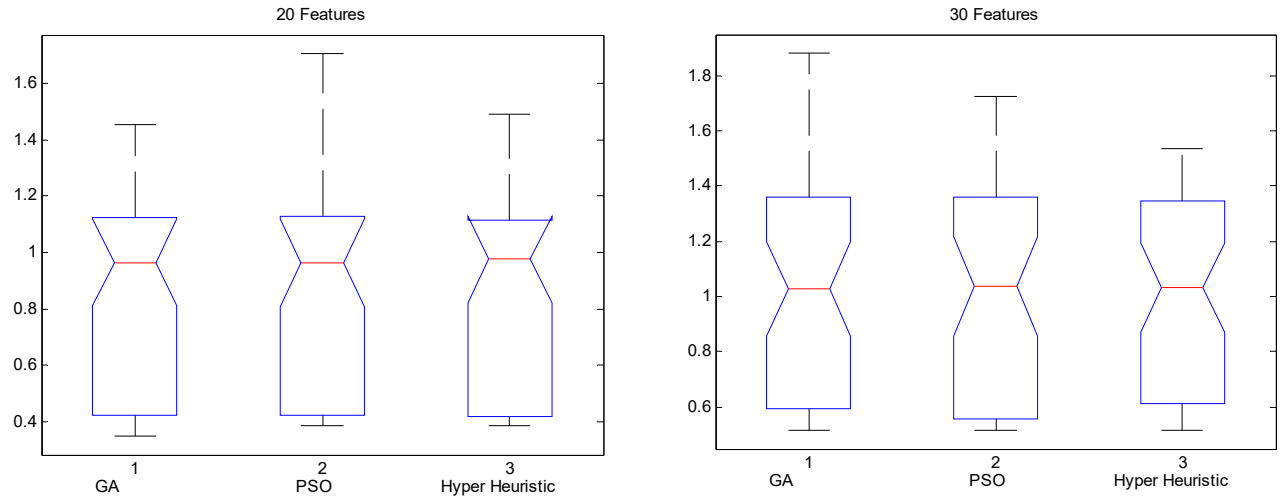
notches of the boxes are not overlapping in all feature sets; therefore, results are significant. Also, PSO has been able to give best feature subset for SPL as compared to genetic algorithm and hyper-heuristic evolutionary algorithm.

Table 5 determines computation time of thirteen different size feature models received by genetic algorithm, particle swarm algorithms and hyper-heuristic evolutionary algorithms. Each algorithm has been run 50 times to compute computation time on feature sets in order to calculate mean and standard deviation. From Table 5 it indicates that the values of computation time of genetic algorithm, PSO, and hyper-heuristic algorithm are closed to each other. The unit of computation time is noted down in seconds. A non-parametric test, i.e., Kruskal-Wallis test has been used and provide p-value. It has been analysed that all reported p-value are above than 0.05 which means that there is no significant difference in computation time among the three algorithms. Figure 3 represents boxplot of computation time of genetic algorithm, PSO, and hyper-heuristic for 20, 30, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, and 1,000 features. In Figure 3, x axis defines values 1, 2 and 3: 1 represents genetic algorithm and 2 represents PSO and 3 represents hyper-heuristic evolutionary algorithm. The y axis defines computation time in seconds. All notches of the boxes are overlapping in Figure 3; therefore, results are not significant, although ranges are varying from each other.

Table 5 Comparison of computation time (seconds) of GA, PSO and hyper-heuristic

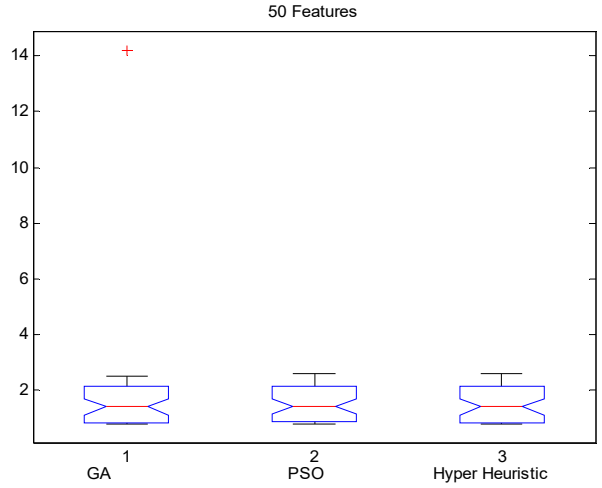
Number of features	*GA*		*PSO*		*Hyper_heuristic*		*p-value*
	Mean	*Standard deviation*	*Mean*	*Standard deviation*	*Mean*	*Standard deviation*	
20	0.8605080	0.348698	0.883874	0.362790	0.854735	0.3309090	9.7070e-01
30	1.028163	0.387779	1.031624	0.380859	0.998499	0.3480100	9.3490e-01
50	1.733894	1.881622	1.509401	0.567989	1.484716	0.5812370	9.4360e-01
100	2.651310	1.046264	2.581146	0.992956	2.647632	1.1359920	8.2150e-01
150	3.664502	1.438492	3.699742	1.357342	3.676034	1.5605370	9.1700e-01
200	4.835783	1.891588	4.882381	1.894673	4.856028	1.9578660	9.5640e-01
250	6.026181	2.509503	5.97586	2.36387	5.827129	2.4279210	8.4030e-01
300	6.996459	2.799624	7.077416	2.845337	6.984392	2.9062500	9.4020e-01
350	8.105071	3.335807	8.065773	3.246564	8.003688	3.262929	9.8820e-01
400	11.23182	15.17490	9.294276	3.843943	9.131215	3.864738	9.8730e-01
450	11.68156	4.989802	11.67331	4.955485	11.658006	5.160042	9.8170e-01
500	11.16248	4.668605	11.17391	4.609803	10.896543	4.499352	8.4880e-01
1,000	22.15224	8.881165	22.50441	9.1577	21.964383	8.757087	7.8080e-01

Figure 3 Boxplot of computation time of GA, PSO and hyper-heuristic, (a) 20 features (b) 30 features (c) 50 features (d) 100 features (e) 150 features (f) 200 features (g) 250 features (h) 300 feature (i) 350 feature (j) 400 features (k) 450 features (l) 500 features (m) 1,000 features (see online version for colours)

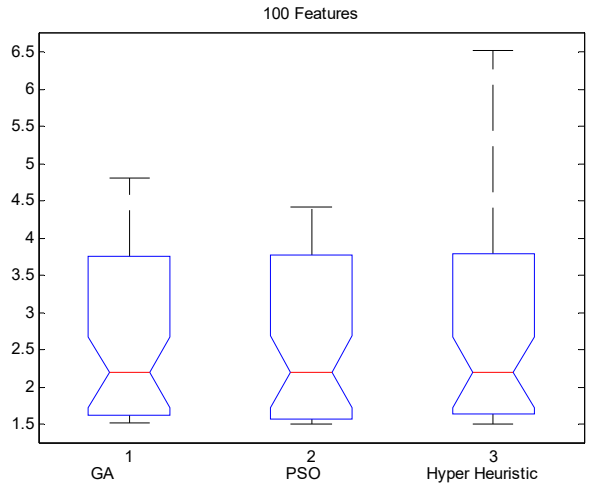


(a) (b)

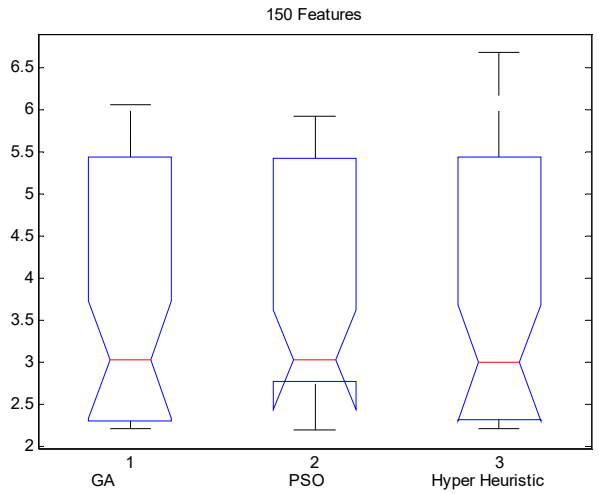
Figure 3 Boxplot of computation time of GA, PSO and hyper-heuristic, (a) 20 features (b) 30 features (c) 50 features (d) 100 features (e) 150 features (f) 200 features (g) 250 features (h) 300 feature (i) 350 feature (j) 400 features (k) 450 features (l) 500 features (m) 1,000 features (continued) (see online version for colours)



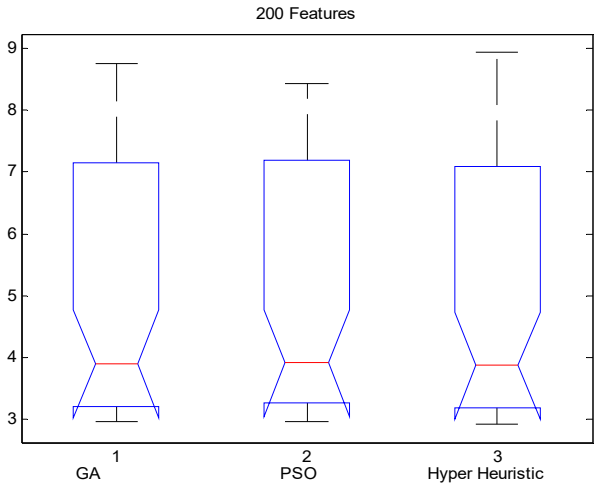
(c)



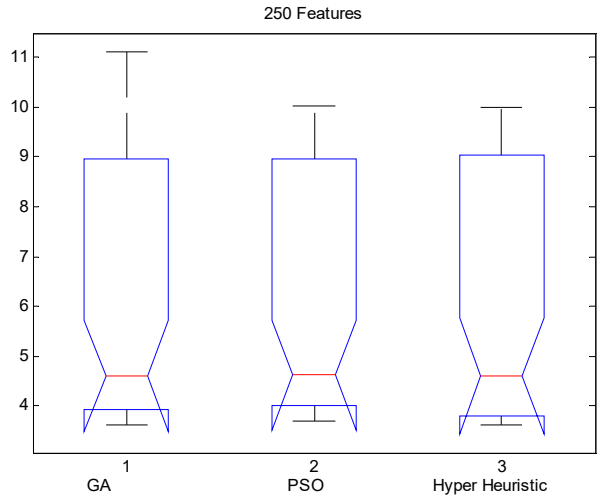
(d)



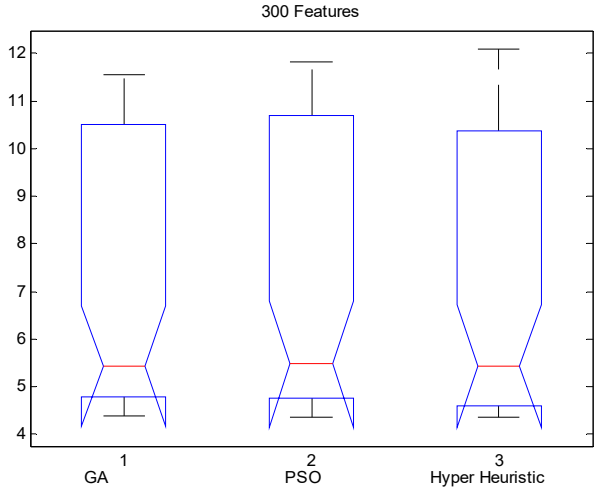
(e)



(f)

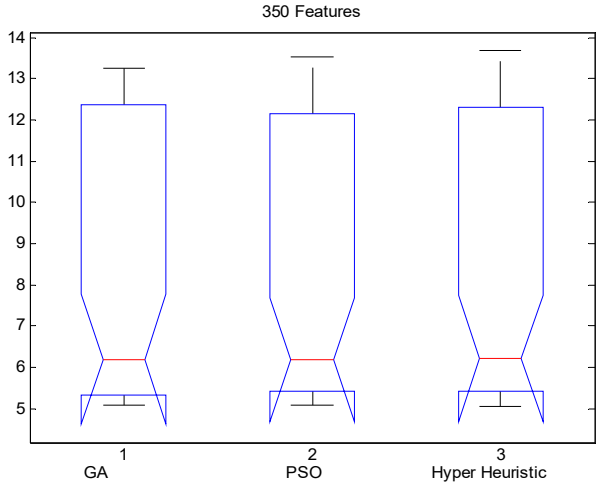


(g)

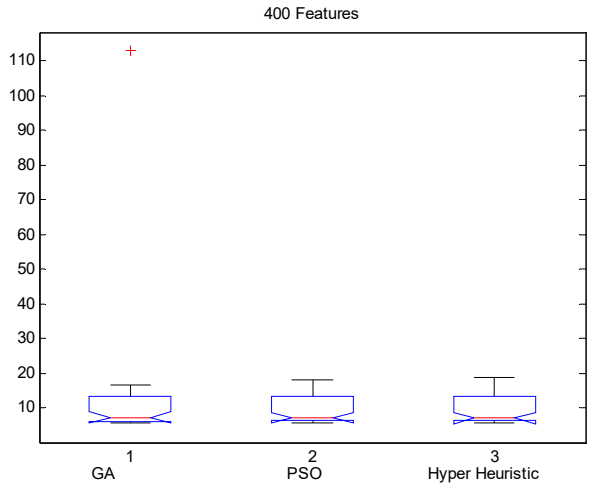


(h)

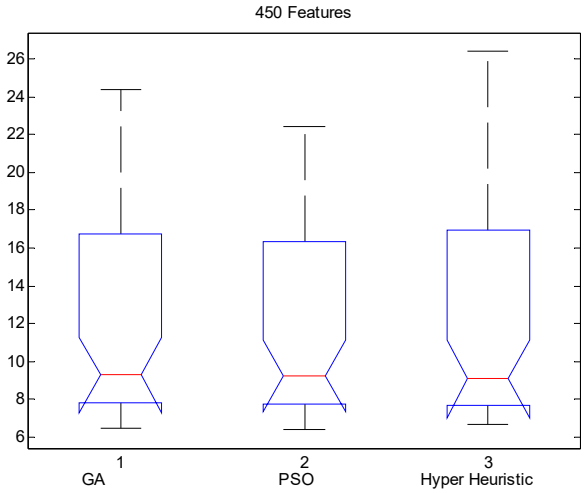
Figure 3 Boxplot of computation time of GA, PSO and hyper-heuristic, (a) 20 features (b) 30 features (c) 50 features (d) 100 features (e) 150 features (f) 200 features (g) 250 features (h) 300 feature (i) 350 feature (j) 400 features (k) 450 features (l) 500 features (m) 1,000 features (continued) (see online version for colours)



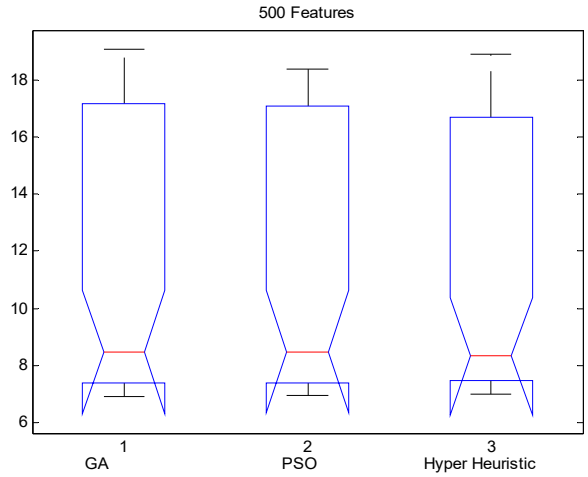
(i)



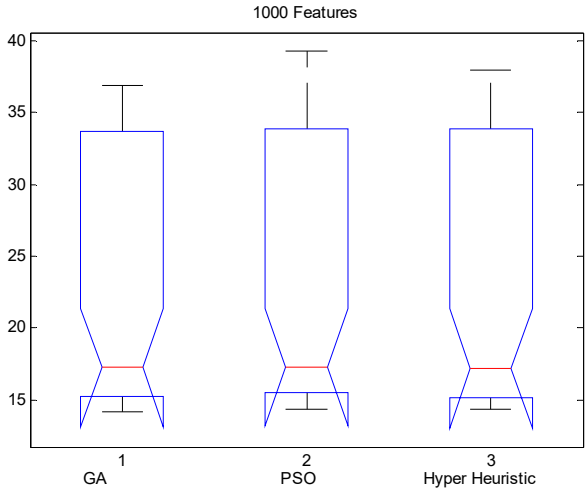
(j)



(k)



(l)



(m)

4 Conclusions and future work

This paper presented GA, PSO and hyper-heuristic to optimise feature selection problem. Every feature has contribution in SPL which can be calculated by reusability and consistency. There are four types of core features I, II, III, IV. Core type I refers to highly reusable features and core type IV refers to least reusable feature. The results show that the analysis will be very useful for industry, especially for those industries where SPLs are used. The genetic algorithm, PSO, and hyper-heuristic have been applied because they do not bind for limited set of features like traditional approaches. In this paper, we optimise multiple feature models from small to large set of feature model using GA, hyper-heuristic and PSO. The three are compared and it is observed that there is no significant difference in computation time and all notches of boxplots are overlapping each other. But in terms of fitness values, there is significant difference among the three algorithms. In boxplots of fitness values, no notches are overlapping each other therefore it shows variation in their range. PSO obtained better results than genetic algorithm and hyper-heuristic evolutionary algorithm. In future, there is a scope for increasing scalability, i.e., addition of new features and deletion of unused features. In order to improve performance, hybridisation of PSO with other metaheuristic algorithm can be done in future.

References

Abbas, A., Wu, Z., Siddiqui, I.F. and Lee, S.U.J. (2016) 'An approach for optimized feature selection in software product lines using union-find and genetic algorithms', *Indian Journal of Science and Technology*, Vol. 9, No. 17, pp.1–8.

Abdelhalim, M.B., Salama, A.E. and Habib, S.E-D. (2006) 'Hardware software partitioning using particle swarm optimization technique,' *6th International Workshop on System on Chip for Real Time Applications*, Cairo, pp.189–194.

Afzal, U., Mahmood, T., Rauf, I. and Shaikh, Z.A. (2014) 'Minimizing feature model inconsistencies in software product lines', *17th IEEE International Multi Topic Conference*, pp.137–142.

Alidra, A. and Kimour, M.T. (2014) 'Towards a software factory for genetic algorithms', *International Journal of Computer and Electrical Engineering*, Vol. 6, No. 1, pp.44–48.

Batory, D. (2005) 'Feature models, grammars, and propositional formulas', *SPLC'05 Proceedings of the 9th international conference on Software Product Lines*, pp.7–20.

Benavides, D., Martin-Arroyo, P.T. and Cortes, A.R. (2005) 'Automated reasoning on feature models', *CAiSE'05 Proceedings of the 17th International Conference on Advanced Information Systems Engineering*, pp.491–503.

Biswas, A., Biswas, B., Kumar, A. and Mishra, K.K. (2018) 'Particle swarm optimisation with time varying cognitive avoidance component', *International Journal of Computational Science and Engineering*, Vol. 16, No. 1, pp.27–41.

Cheng, S., Chen, J., Qin, Q. and Shi, Y. (2018) 'Population diversity of particle swarm optimisation algorithms for solving multimodal optimisation problems', *International Journal of Computational Science and Engineering*, Vol. 17, No. 1, pp.69–79.

Chhikara, R.R., Sharma, P. and Singh, L.(2016) 'A hybrid feature selection approach based on improved PSO and filter approaches for image steganalysis', *International Journal of Machine Learning and Cybernetics*, Springer, Vol. 7, No 6, pp.1195–1206.

Chohan, A.Z., Bibi, A. and Motla, Y.H. (2017) 'Optimized software product line architecture and feature modeling in improvement of SPL', *International Conference on Frontiers of Information Technology (FIT)*, pp.167–172.

Cowling, P., Kendall, G. and Soubeiga, E. (2001) 'Hyperheuristic approach to scheduling a sales summit', *Proceedings of the Third International Conference of Practice and Theory of Automated Timetabling*, pp.176–190.

Czarnecki, K. and Wasowski, A. (2007) 'Feature diagrams and logics:there and back again', *11th International Software Product Line Conference*, pp.23–34.

Ding, J., Hao, K. and Hou, H. (2011) 'The research on measurement and management of core asset library', *2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*, pp.3542–3545.

Eberhart, R. and Kennedy, J. (1995) 'A new optimizer using particle swarm theory', *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, pp.39–43.

Engelbrecht, A.P. (2007) *Computation Intelligence: An Introduction*, 2nd ed., University of Pretoria, South Africa.

Ferreira, T.N., Lima, J.A.P., Strickler, A., Kuk, J.N., Vergilio, S.R. and Pozo, A. (2017) 'Hyper-heuristic based product selection for software product line testing', *IEEE Computational Intelligence Magazine*, Vol. 12, No. 2, pp.34–45.

Filho, H.L.J., Ferreira, T.N. and Vergilio, S.R. (2018) 'Incorporating user preferences in a software product line testing hyper-heuristic approach', *IEEE Congress on Evolutionary Computation (CEC)*, pp.1–8.

Guo, J., Liang, J.H., Shi, K., Yang, D., Zhang, J., Czarnecki, K., Ganesh, V. and Yu, H. (2019) 'SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines', *Software & Systems Modeling*, Vol. 18, No. 2, pp.1447–1466.

Guo, J., White, J., Wang, G., Li, J. and Wang, Y. (2011) 'A genetic algorithm for optimized feature selection with resource constraints in software product lines', *Journal of Systems and Software*, Vol. 84, No. 12, pp.2208–2221.

Henard, C., Papadakis, M., Perrouin, G., Klein, J. and Le Traon, Y. (2013) 'Multi-objective test generation for software product lines', *Proceedings of the 17th International Software Product Line Conference on – SPLC*, pp.62–71.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA.

Kumar, S. and Rajkumar (2017) 'Cost-based test case prioritization technique for software product line', *International Journal of Scientific Progress and Research*, Vol. 40, No. 4, pp.206–212.

Kumari, A.C. (2018) 'Feature selection optimization in SPL using genetic algorithm', *Procedia Computer Science*, Vol. 132, pp.1477–1486.

Kumari, A.C. and Srinivas, K. (2016) 'Hyper-heuristic approach for multi-objective software module clustering', *Journal of Systems and Software*, Vol. 117, pp.384–401.

Kumari, A.C., Srinivas K. and Gupta M.P. (2013) 'Software module clustering using a hyperheuristic based multi-objective genetic algorithm', *3rd IEEE International Advance Computing Conference (IACC)*, pp.813–818.

Li, J., Liu, X., Wang, Y. and Guo, J. (2011) 'Formalizing feature selection problem in software product lines using 0-1 programming', in *6th International Conference on Intelligent Systems and Knowledge Engineering*, pp.459–465.

Linsbauer, L., Lopez-Herrejon, R.E. and Egyed, A. (2014) 'Feature model synthesis with genetic algorithm', *Search-Based Software Engineering: 6th International Symposium*, pp.153–167.

Mamun, A.A., Djatmiko, F. and Das, M.K. (2016) 'Binary multi-objective PSO and GA for adding new features into an existing product line', *19th International Conference on Computer and Information Technology (ICCIT)*, pp.581–585.

Saed, A.A.A. and Kadir, W.M.N.W. (2011) 'Applying particle swarm optimization to software performance prediction an introduction to the approach', *Malaysian Conference in Software Engineering*, pp.207–212.

Sahin, F. (2014) 'A survey on feature selection methods', *Computers and Electrical Engineering*, Vol. 40, No.1, pp.16–28.

Saka, M. and Dogan, E. (2012) 'Recent developments in metaheuristic algorithms: a review', *Computational Technology Reviews*, Vol. 5, pp.31–78.

Sayyad, A.S., Ingram, J., Menzies, T. and Ammar, H. (2013) 'Optimum feature selection in software product lines: let your model and values guide your search', *1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, San Francisco, CA, pp.22–27.

Sharma, D.K., Hitesh and Rao, V. (2014) 'Configurable business process modeling notation', *IEEE International Advance Computing Conference (IACC)*, pp.1424–1429.

Sharma, D.K., Hitesh and Rao, V. (2015) 'Individualization of process model from configurable process model constructed in C-BPMN', *International Conference on Computing, Communication and Automation, ICCCA 2015*, pp.750–754.

Shi, Y. and Eberhart, R. (1998) 'A modified particle swarm optimizer', *Proceedings of the IEEE World Congress on Computational Intelligence*, pp.69–73.

Sushama, C. and Reddy, A.R.M. (2018) 'An automation approach for architecture discovery in software design using genetic algorithm', *International Journal of Computational Science and Engineering*, Vol. 17, No. 4, pp.390–397.

Tsafarakis, S., Marinakis, Y. and Matsatsinis, N. (2011) 'Particle swarm optimization for optimal product line design', *International Journal of Research in Marketing*, Vol. 28 No.1, pp.13–22.

Tu, C-J., Chuang, L-Y., Chang, J-Y. and Yang, C-H. (2006) 'Feature selection using PSO-SVM', *IAENG International Journal of Computer Science*, Vol. 33, No. 1.

Wang, Y.L. and Pang, J.W. (2014) 'Ant colony optimization for feature selection in software product lines', *Journal of Shanghai Jiaotong University (Science)*, Vol. 19, No. 1, pp.50–58.

White, J., Dougherty, B. and Schmidt, D.C. (2009), 'Selecting highly optimal architectural feature sets with filtered Cartesian flattening', *Journal of Systems and Software*, Vol. 82, No. 8, pp.1268–1284.

Wu, C., Lai, K. and Sun, R. (2008) 'GA-based job scheduling strategies for fault tolerant grid systems', *2008 IEEE Asia-Pacific Services Computing Conference*, Yilan, pp.27–32.

Yadav, H. and Kumari, A.C. (2018) 'Analysis of features using feature model in software product line: a case study', *International Journal of Education and Management Engineering (IJEME)*, Vol. 8, No. 2, pp.48–57.