Hybridizing PSO with SA for Optimizing SVR Applied to Software Effort Estimation

Dinda Novitasari, Imam Cholissodin, Wayan Firdaus Mahmudy
Dept. of Informatics/ Computer Science, Brawijaya University, Malang 65145
e-mail: id.dindanovitasari@gmail.com, imamcs@ub.ac.id, wayanfm@ub.ac.id

Abstract
This study investigates Particle Swarm Optimization (PSO) hybridization with Simulated Annealing (SA) to optimize Support Vector Machine (SVR). The optimized SVR is used for software effort estimation. The optimization of SVR consists of two sub-problems that must be solved simultaneously; the first is input feature selection that influences method accuracy and computing time. The next sub-problem is finding optimal SVR parameter that each parameter gives significant impact to method performance. To deal with a huge number of candidate solutions of the problems, a powerful approach is required. The proposed approach takes advantages of good solution quality from PSO and SA. We introduce SA based acceptance rule to accept new position in PSO. The SA parameter selection is introduced to improve the quality as stochastic algorithm is sensitive to its parameter. The comparative works have been between PSO in quality of solution and computing time. According to the results, the proposed model outperforms PSO SVR in quality of solution.

Keywords: Particle swarm optimization, simulated annealing, support vector regression, feature selection, parameter optimization

Copyright © 2016 Universitas Ahmad Dahlan. All rights reserved.

1. Introduction
The most important part of software project is software effort estimation. It determines how many resources that project needed and must be done accurately. If we have big error rate in estimation, it will lead into big loss such as unpredictable delay time and unexpected budget. To prevent many losses in the future, some approaches are developed to estimate software effort. One of them is machine learning. Support vector machine is machine learning algorithm introduced by Vapnik to solve classification problem. Due to solve real world problems, SVM was developed to solve regression and time series prediction called SVM based regression (SVR). In order to solve nonlinear regression problem, SVR mapped data to high dimensional feature space using kernel function. This kernel must satisfy Mercer condition [1] and one of kernels is radial basis function (RBF).

In machine learning, feature selection introduced as a process of selecting a subset feature for use in model construction. This process is needed for SVR since it can simplify computing process and reducing computing time, especially when computing in high dimensional space. Besides that, proper parameter settings can influence SVR accuracy. SVR-RBF has parameters influenced its performance i.e. error penalty, insensitive loss, and radial basis [2]. Those mentioned above are crucial in SVR-RBF because feature selection influences SVR parameter and vice versa [3]. In the past research, Oliviera investigated the use of SVR in order to do software effort estimation [4]. It gives promising result but cannot guarantee give good result since using predefined number of features and parameter means cannot discover other options that can lead into higher accuracy rate. Numerous candidate of solution can be generated in order to have great number of subset feature combination and vary range of parameter, if we use enumeration. However, it does not utilize a fitness function, and is thus unguided, often failing to find good solution. Due to the complexity of the problem, a powerful approach is required to get a good solution.

Some stochastic optimization methods become alternative to select subset feature and optimize parameter. It generates candidate solutions, involves objective function to evaluate the quality of solution so solution searching could be lead into a good solution. Braga et al proposed genetic algorithm (GA) to optimize SVR in software effort estimation [5]. Our previous research...
proposed particle swarm optimization (PSO) to optimize SVR in the same problem domain [6]. Basically, PSO is inspired by flocking bird motion employed parallel search techniques, exploitation and exploration. However, PSO has disadvantage, trapped in local minimum because particles move in high velocity and gain premature convergence [7]. On the other hand, simulated annealing inspired by process of annealing in metallurgy, is good in finding local optimum [8]. Therefore, this study investigates hybridization PSO with SA in order to enhance searching capacity. This proposed model is used to optimize SVR parameter and select subset feature applied to software effort estimation.

Several researches investigated SVR optimization have been done and gained promising result. Braga et.al [5] investigated GA application to select subset feature and SVR parameter applied to software effort estimation. They used binary coded chromosome as solution representation for subset feature and SVR parameter. Their research reported success to improve SVR performance. Our previous research [6] investigated PSO application to select subset feature and SVR parameter applied to software effort estimation. We used continuous value type to optimize SVR parameter and discrete value type to select subset feature. Another effort has been done by Adhani [9], who optimized SVR with GAPSO. They are reported success to build SVR model for predicting rainfall in dry season. However, other researches have been done to investigate on how to improve PSO performance. Xue [10] introduced QoS-based hybrid particle swarm optimization (GHPSO) to schedule workflow in cloud computing. It gained better performance than PSO. A research conducted by Shieh et.al [11] in modification PSO with SA. Their research proposed SAPSO to enhance searching capacity algorithm. They reported proposed model could have higher efficiency, better quality and faster convergence than PSO. Therefore, based on past researches, this study proposed SAPSO SVR applied to software effort estimation. By using SAPSO, can be generated more optimize SVR parameter, better selected feature, and low cost value.

2. Support Vector Regression

Given training data \( \{x_i, y_i\}, i = 1,\ldots,l; x_\in R^d; y_\in R^d \) where \( x_i, y_i \) is input (vector) and output (scalar value as target). Other forms of alternative for bias to calculation \( f(x) \) is can be build solution like bias as follows [1]:

\[
b = y_i - \varepsilon - w^T \Phi(x_i)
\]

\[
= y_i - \varepsilon - \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) (x_i \cdot x_i)
\]

\[
= y_i - \varepsilon - \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) k(x_i, x_i)
\]

\( x_i \) is support vector where \( [\alpha_i - \alpha_i^*] \) isn’t zero. Equation \( f(x) \) can be written as follows:

\[
f(x) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) (x \cdot x_i) + b
\]

\( \lambda \) is scalar constant, with it’s an augmented factor defined as follows [12]:

\[
f(x) = \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) (K(x_i, x) + \lambda^2).
\]

2.1. Sequential Algorithm for SVR

Vijayakumar has made tactical steps through the process of iteration to obtain the solution of optimization problems of any nature by way of a trade-off on the values of the weights \( x_i \), or called \( \alpha_i \) to make the results of the regression becomes closer to actual value. The step by step as follows [12]:

1. Initialize \( \alpha_i = 0, \alpha_i^* = 0 \). Compute \( \left[R_{ij} = K(x_i, x_j) + \lambda^2 \right. \)

for \( i,j = 1,\ldots,n \)

2. For each training point \( x_i \), \( i=1 \) to \( n \), compute:
\[
E_i = y_i - \sum_{j=1}^{n} (\alpha^*_i - \alpha_i)R_y
\]

(5)

\[
\delta \alpha_i^* = \min \{ \max \{ \gamma (E_i - \varepsilon), -\alpha_i^* \}, C - \alpha_i^* \}.
\]

(6)

\[
\delta \alpha_i = \min \{ \max \{ \gamma (-E_i - \varepsilon), -\alpha_i \}, C - \alpha_i \}.
\]

(7)

\[
\alpha_i^* = \alpha_i + \delta \alpha_i^*.
\]

(8)

3. Repeat step 2 until meet stop condition.

Where learning rate \(\gamma\) is computed from:

\[
\omega = \frac{\text{learning rate constant}}{\text{max (diagonal of kernel matrix)}}
\]

(10)

3. Particle Swarm Optimization

Particle swarm optimization was introduced by Kennedy and Ebenhart [13], as a nature inspired algorithm. Particles are defined as solution for problem. Developing by Shi and Ebenhart [14], PSO is added by inertia weight to improves performance. Each particle has position and velocity, and updates that in every iterating. The velocity is updated by:

\[
v_{ij}(t+1) = w_{ij}(t) + c_1 r_1(t) [y_{ij}(t) - x_{ij}(t)] + c_2 r_2(t) [\hat{y}(t) - x_{ij}(t)]
\]

(15)

And its position updated by:

\[
x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1)
\]

(16)

Where \(v_{ij}(t)\) is velocity of particle \(i\) in dimension \(j=1,...,n\) at time \(t\). \(x_{ij}(t)\) is position of particle \(i\) in dimension \(j\) at time \(t\). \(c_1\) and \(c_2\) are acceleration constants used to scale contribution of the cognitive and social components, \(r_1\) and \(r_2\) are random values in the range \([0,1]\). \(W\) is inertia weight obtained by:

\[
\omega = (\omega_{\text{max}} - \omega_{\text{min}}) \frac{\text{iter}_{\text{max}} - \text{iter}}{\text{iter}_{\text{max}}} + \omega_{\text{min}}
\]

(17)

Where \(w_{\text{max}}\) and \(w_{\text{min}}\) are maximal and minimum inertia weight, \(\text{iter}_{\text{max}}\) is maximum number of iterations, \(\text{iter}\) is current iteration number. \(y_i\) is personal best position of particle \(i\) obtained by:

\[
y_i(t+1) = \begin{cases} 
y_i(t) & \text{if } f(y_i(t+1)) \geq f(y_i(t)) \\
x_i(t+1) & \text{if } f(y_i(t+1)) < f(y_i(t))
\end{cases}
\]

(18)

And \(\hat{y}\) represents global best position of particle \(i\) obtained by:

\[
\hat{y}(t) \in \{y_0(t),...,y_{ns}(t)\} | f(\hat{y}(t)) = \min \{ f(y_0(t)),..., f(y_{ns}(t)) \}
\]

(19)

3.1. Binary PSO

Some optimization problems are set in a space featuring discrete. Kennedy and Ebenhart [15] proposed binary PSO in which each element of particle’s position vector can take on the binary value 0 or 1. New velocity of particle is normalized by sigmoid function:
\[ v_i^j(t) = \text{sig}(v_i^j(t)) = \frac{1}{1 + e^{-v_i^j(t)}} \]  

(20)

Where \( v_i^j(t) \) is obtained from Equation (15). Using Equation (16), the position update changes to:

\[
x_i^j(t+1) = \begin{cases} 
1 & \text{if } r_{ij}(t) < \text{sig}(v_i^j(t+1)) \\
0 & \text{otherwise}
\end{cases}
\]

(21)

Where \( r_{ij}(t) \sim U(0,1) \).

4. Hybridizing PSO with SA

A searching algorithm has two important components, exploration and exploitation. Exploration means algorithm search in different region of searching space to find global optimum. Exploitation means algorithm localize promising area to find best solution in that area. A good searching algorithm must able to balance its exploration and exploitation, able to search entire space and jump out of local optimum solution. By that means, it must able to improve probability and ability of finding global optimum solution.

Initial random position PSO can lead into premature convergence, entire particle move toward local optimum solution and cause weakening exploration because particle can’t jump out of area. It is characteristic and weakness of PSO. Meanwhile, SA with low variation of temperature parameter and searching solution reach equilibrium condition, able to guarantee to find global optimum. It is enhanced by metropolis process, ability to jump out from local optimum. However, it costs high computing time.

Based on PSO and SA characteristic above, this study hybridizes PSO with SA, combines PSO parallel process and movement mechanism and SA searching procedure. By combining that, this proposed model able to find good solution in local and global optimum with low computing time.

4.1. Simulated Annealing Algorithm

Simulated annealing is an optimization process based on the annealing process; the cooling process of a liquid or solid and the analysis of the behavior of substances as they cool. This algorithm is introduced by Kirkpatrick [16] and inspired by Metropolis work about energy distribution [17]. In SA algorithm, metropolis process does searching solution. During the process, disturbance mechanism (metropolis acceptance rule) determines quality of solution by searching around existing solution and comparing neighbor solution and current solution. This procedure affects SA ability to jump out from local optimum solution. If neighbor solution is better than current solution then neighboring solution is accepted as the new current solution. If neighbor solution is worse than current solution then SA will use a probability to determine whether accept this neighboring solution as new current solution or not, or regenerate for a new neighboring solution. The probability mechanism for metropolis acceptance rule is defined as follows:

\[
P = \begin{cases} 
1 & \text{if } f(x_j) < f(x_i) \\
\frac{e^{-\frac{f(x_j) - f(x_i)}{kT}}}{P} & \text{otherwise}
\end{cases}
\]

(22)

Where \( P \) is probability, \( f(x_j) \) is neighbor solution, \( f(x_i) \) is current solution, \( c_0 > 0 \) is Boltzmann constant and \( T \) is temperature of the system. \( T \) is derived from:

\[
T_{k+1} = \alpha \times T_k
\]

(23)

Where \( \alpha \) is cooling rate, \( T_{k+1} \) is temperature at time \( k \), and \( T_0 \) is initial temperature. While SA is quite simple, it has been successfully implemented to solve various combinatorial problem [18].
5. SAPSO SVR Model
5.1. Particle Representation
In this study, SVR RBF is defined by the parameter $C$ – complexity parameter, $\varepsilon$ - the extent to which deviations are tolerated, $\lambda$ - augmenting factor, $\sigma$ – width of RBF kernel, $cLR$ – learning rate constant. The particle is comprised of six parts: $C$, $\varepsilon$, $\lambda$, $\sigma$, $cLR$ (continuous-valued) and features mask (discrete-valued). Table 1 shows the representation of particle $i$ with dimension $n_f+5$ where $n_f$ is the number of features. The feature mask is Boolean that “1” indicates the feature is selected and “0” indicates feature is not selected.

Table 1. Particle $i$ is composed of six parts: $c$, $\varepsilon$, $\lambda$, $\sigma$, $cLR$ and feature mask

<table>
<thead>
<tr>
<th>Continuous-valued</th>
<th>Discrete-valued</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>Feature mask</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td></td>
</tr>
<tr>
<td>$\lambda$</td>
<td></td>
</tr>
<tr>
<td>$\sigma$</td>
<td></td>
</tr>
<tr>
<td>$cLR$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$X_{i,1}$</th>
<th>$X_{i,2}$</th>
<th>$X_{i,3}$</th>
<th>$X_{i,4}$</th>
<th>$X_{i,5}$</th>
<th>$X_{i,6}$</th>
<th>$X_{i,7},...,X_{i,n_f}$</th>
</tr>
</thead>
</table>

5.2. Objective Function
Objective function is used to measure how optimal the generated solution. There are two types of objective function: fitness and cost. The higher fitness value means better solution. The lower cost value means better solution. In this study, cost typed is used as objective function because the purpose of this algorithm is to minimize error. Accuracy of prediction and number of selected features are criteria used to design cost function. Thus, the particle with high accuracy of prediction and small number of features produces a low prediction error. The prediction error has two predefined weights: $W_A$ for accuracy of prediction (95%) and $W_F$ for the selected feature (5%) [19].

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{A_i - F_i}{A_i} \right|$$

(24)

$$\text{error} = \left( W_A \times \text{MAPE} \right) + \left( W_F \times \frac{\sum_{j=1}^{n_f} f_j}{n_F} \right)$$

(25)

Where $n$ is number of data, $A_i$ is actual value and $F_i$ is prediction value for data, $f_j$ is value of feature mask where “1” represents that feature $j$ is selected and “0” represents that feature $j$ is not selected and $n_f$ is total number of features.

5.3. SAPSO SVR Algorithms
The SAPSO SVR algorithm is started by initialization of particle. Then, calculate cost and determine personal best position ($pBest$) and global best position ($gBest$). After that, update velocity and position. Usually, PSO automatically accept new position, however SAPSO SVR introduces SA metropolis acceptance rule in this step. This rule determines whether to accept new position or regenerate another candidate position based on cost function difference between new and old positions. This enables PSO to jump out from local optimum, improve quality of solution, and increase rate of convergence. Simulated annealing explores solution towards direction of $pBest$ and $gBest$. The acceptance rule accepts or rejects new solution based on current temperature parameter and cost value difference. If candidate solution unable pass criteria then a new position generated using PSO and repeated until metropolis acceptance rule accept new position or upper bound of disturbance is reached. By this way, the model explores solution, improve exploration and spend low computing time since using PSO parallel processing.

Based on Figure 1, the whole procedure of SAPSO SVR is described as follows:
1. Normalizing data using

$$X_n = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

(26)
Where $x$ is the original data from dataset, $x_{\text{min}}$ and $x_{\text{max}}$ is the minimum and maximum value of original data, and $x_n$ is normalized value.

2. Dividing data into $k$ to determine training and testing data.
3. Initializing a population of particle randomly $s_{i0}, v_{i0} = 0$ for $i = 1, 2, \ldots, n, \text{iter} = 0$.
4. Calculating cost of $s_{i0}$ by averaging error over $k$ SVR training.
5. Updating $pBest$ and $gBest$ of each particle.
6. Updating inertia weight.
7. Repeat these steps until meet stopping condition
   a) Updating velocity $v_{i}^{\text{iter}+1}$ and position of each particle.
   b) Calculating cost of $s_{i}^{\text{iter}+1}$ by averaging error over $k$ SVR training.
   c) Evaluate $\Delta\text{cost} = \text{cost}(s_{i}^{\text{iter}+1}) - \text{cost}(s_{i}^{\text{iter}})$ and generate random number $R \in [0,1]$. If $\Delta\text{cost} \leq 0$, then accept new position with probability ONE. Otherwise, $s_{i}^{\text{iter}+1}$ is accepted based on following criterion: $\text{probability} = e^{-\frac{\Delta\text{cost}}{\text{temperature}}} \geq R$. Proceed to next step if all new positions are accepted or repeat step 7.1 until 7.3 for those particles failed to be accepted. Too many failures (i.e. 100 in our study) for same particle will force the last position will be accepted.
   d) Updating $pBest$ and $gBest$ of each particle.
   e) Updating inertia weight and temperature, set $\text{iter} = \text{iter} + 1$.
8. If stopping criteria is satisfied, and then end iteration. If not, repeat step 7. In this study, stopping criteria is a given number of iterations.
9. Output the best solution $gBest$ and its cost value.

Figure. 1 Flowchart of SAPSO SVR algorithm
6. Application SAPSO SVR in Software Effort Estimation
6.1. Simulation Settings

This study simulates 2 algorithms: PSO SVR and SAPSO SVR programmed using C#. For SAPSO SVR simulation, we use the same parameter and dataset that is obtained from [6] that conducted PSO SVR simulation. For software effort estimation, the inputs of SVR are Desharnais dataset [20]. The Desharnais dataset consists of 81 software projects described by 11 variables, 9 independent variables and 2 dependent variables. For the simulation, we decide to use 77 projects due to incomplete provided features and 7 independent variables (TeamExp, ManagerExp, Transactions, Entities, PointsAdjust, Envergure, and PointsNonAdjust) and 1 dependent variable (Effort). The PSO parameters were set as in Table 2. Firstly, we run test to determine best parameter for SA (T₀ and α) then simulations is performed and compared to other algorithms.

<table>
<thead>
<tr>
<th>Number of fold</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population of particles</td>
<td>20</td>
</tr>
<tr>
<td>Number of iterations</td>
<td>40</td>
</tr>
<tr>
<td>Inertia weight (w_max, w_min)</td>
<td>(0.9, 0.4)</td>
</tr>
<tr>
<td>Acceleration coefficient (c₁, c₂)</td>
<td>(2, 2)</td>
</tr>
<tr>
<td>Parameter searching space</td>
<td>C (0.1-1500), ε (0.001-0.009), σ (0.1-4), λ (0.01-1.3), cLR (0.01-1.75)</td>
</tr>
</tbody>
</table>

6.2. Best Parameter

In stochastic algorithms, parameters have effect to quality of generated solution. In SA, initial temperature and cooling rate influence its performance. By observing parameters, we choose best parameter has lowest cost in each simulation.

Table 3 showed simulation to choose best initial temperature (T₀). This simulation conducted by increasing temperature by 10% from 50 up to 90 in each simulation and use cooling rate at 0.5. If T₀ is too low then algorithm has possibility to not explore, makes converge at local optimum. If T₀ is too high, then it can increase computing time. This table showed that T₀ at 90 give lowest cost.

Table 4 showed simulation to choose best cooling rate (α). This simulation conducted by increasing temperature by 10% from 0.5 up to 0.9 in each simulation If α is too low then algorithm has possibility to fail into local optimum solution, repeats calculation, and increase computing time. If α is too high, then it increase computing time. This table showed that α at 0.9 give lowest cost.

<table>
<thead>
<tr>
<th>i-th simulation</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6084</td>
<td>0.8336</td>
<td>0.5801</td>
<td>0.6096</td>
<td>0.5690</td>
</tr>
<tr>
<td>2</td>
<td>0.5706</td>
<td>0.5760</td>
<td>0.7265</td>
<td>0.5975</td>
<td>0.5734</td>
</tr>
<tr>
<td>3</td>
<td>0.5992</td>
<td>0.5898</td>
<td>0.5886</td>
<td>0.5848</td>
<td>0.5682</td>
</tr>
<tr>
<td>4</td>
<td>0.5610</td>
<td>0.6177</td>
<td>0.6223</td>
<td>0.5875</td>
<td>0.5935</td>
</tr>
<tr>
<td>5</td>
<td>0.7475</td>
<td>0.7579</td>
<td>0.6161</td>
<td>0.5795</td>
<td>0.5761</td>
</tr>
<tr>
<td>Average Cost</td>
<td>0.6174</td>
<td>0.6750</td>
<td>0.6267</td>
<td>0.5918</td>
<td>0.5760</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i-th simulation</th>
<th>α</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5690</td>
<td>0.5861</td>
<td>0.5776</td>
<td>0.5769</td>
<td>0.5659</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.5734</td>
<td>0.5957</td>
<td>0.5994</td>
<td>0.6163</td>
<td>0.5563</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.5682</td>
<td>0.5898</td>
<td>0.6182</td>
<td>0.6003</td>
<td>0.5748</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.5935</td>
<td>0.5765</td>
<td>0.6007</td>
<td>0.5753</td>
<td>0.5575</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.5761</td>
<td>0.6013</td>
<td>0.6026</td>
<td>0.5778</td>
<td>0.5628</td>
<td></td>
</tr>
<tr>
<td>Average Cost</td>
<td>0.5760</td>
<td>0.5899</td>
<td>0.5997</td>
<td>0.5893</td>
<td>0.5635</td>
<td></td>
</tr>
</tbody>
</table>
6.3. Comparison Works

By using best parameter, we compare SAPSO SVR and PSO SVR performance. Figure 2 showed comparison of convergence between PSO SVR and SAPSO. It showed that SAPSO have faster convergence than PSO SVR. In Table 5, we can see that SAPSO has higher computing time faster than PSO SVR but on the other hand, SAPSO also can have faster convergence and lower cost than PSO SVR. The error difference of error is big, but the high computing time can be compromised. The computing time is high because the model must repeat searching candidate position if they fail meet the acceptance rule criteria and this is different with PSO automatically accept candidate position.

Table 5. Simulation result

<table>
<thead>
<tr>
<th>Model</th>
<th>Time (ms)</th>
<th>Optimal (C, ε, σ, cLR, λ)</th>
<th>Selected features</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO-SVR</td>
<td>50238</td>
<td>393,04, 0,09, 0,1, 0,01, 1,6669</td>
<td>2 (PointsAdjust and PointsNonAdjust)</td>
<td>0,5981</td>
</tr>
<tr>
<td>SAPSO-SVR</td>
<td>83756</td>
<td>1500, 0,0401, 0,3996, 0,01, 0,01</td>
<td>2 (Envergure, and PointsNonAdjust)</td>
<td>0,5575</td>
</tr>
</tbody>
</table>

Figure 2. Convergence curve PSO SVR and SAPSO

7. Conclusion

This study investigated the use of SAPSO for optimal feature subset selection and SVR parameters optimization in the problem of software effort estimation. In our simulations, we used Desharnais dataset. We compared our results to PSO-SVR. From the experiment results, using SA can improve performance of PSO. The proposed model can combine the advantage of both algorithms and gain lower cost than PSO.

References


