LSSTCS: An Efficient Particle Swarm Optimization for Large-Scale Hardware/Software Co-Design System

Xiaohu Yan, Fazhi He*, Neng Hou and Haojun Ai

State Key Laboratory of Software Engineering
Wuhan University, Wuhan 430072, P. R. China

School of Computer Science
Wuhan University, Wuhan 430072, P. R. China

*Corresponding author.

Received 25 December 2016
Revised 20 February 2017
Accepted 23 March 2017
Published 27 April 2017

In the co-design process of hardware/software (HW/SW) system, especially for large and complicated embedded systems, HW/SW partitioning is a challenging step. Among different heuristic approaches, particle swarm optimization (PSO) has the advantages of simple implementation and computational efficiency, which is suitable for solving large-scale problems. This paper presents a conformity particle swarm optimization with fireworks explosion operation (CPSO-FEO) to solve large-scale HW/SW partitioning. First, the proposed CPSO algorithm simulates the conformist mentality from biology research. The CPSO particles with psychological conformist always try to move toward a secure point and avoid being attacked by natural enemy. In this way, there is a greater possibility to increase population diversity and avoid local optimum in CPSO. Next, to enhance the search accuracy and solution quality, an improved FEO with new initialization strategy is presented and is combined with CPSO algorithm to search a better position for the global best position. This combination can keep both the diversified and intensified searching. At last, the experiments on benchmarks and large-scale HW/SW partitioning demonstrate the efficiency of the proposed algorithm.

Keywords: Hardware/software partitioning; particle swarm optimization; fireworks explosion operation; communication cost; parallel computing.

1. Introduction

Co-design is one area of cooperative information process, system and application (CIPSA). Optimization approach is necessary in CIPSA. An efficient optimization approach is very important for large-scale CIPSA.

Hardware/software (HW/SW) partitioning is a key step in HW/SW co-design system. HW/SW partitioning decides which tasks of the system should be implemented in hardware and which ones in software. Implementation with software module has more flexibility and needs less cost, but more executing time, and vice...
versa in hardware case. The target of HW/SW partitioning is to balance all the
tasks to optimize some objectives of the system under some constraints.

In early studies, the HW/SW partitioning problem has specific optimization
objectives and the main HW/SW partitioning methods are exact algorithms,
such as dynamic programming, branch-and-bound (B&B) and integer linear
programming (ILP). As most formulations of the partitioning problem are
NP-hard, exact algorithms cannot provide feasible solution for large-scale HW/SW
partitioning problems. Thus, many researchers have applied heuristic algorithms
to HW/SW partitioning, such as genetic algorithm (GA), simulated annealing
(SA), greedy algorithms, tabu search (TS), ant colony optimization (ACO) and particle swarm optimization (PSO). All these heuristic
approaches can work perfectly within their own design environments, but due to
the enormous differences among them, it is not possible to compare the results
obtained.

With the rapid development of the design complexity in embedded systems, the
scale of HW/SW partitioning problems becomes extremely large. PSO may easily
get trapped into local optimum for large-scale HW/SW partitioning. In this paper,
conformity particle swarm optimization with fireworks explosion operation (CPSO-
FEO) is proposed to solve large-scale HW/SW partitioning problems. To enhance
the searching diversification and avoid local optimum, the particles in CPSO-FEO
move towards the secure point where there are lots of particles and there is less
possibility to be attacked by predator. To enhance the searching intensification and
the solution quality, fireworks explosion operation is integrated to update the global
best position in the population. It has been proved that heuristic algorithms take
too much time in addressing large-scale HW/SW partitioning problems. Hence, it is
very necessary to reduce the runtime of heuristic algorithms. To accelerate HW/SW
partitioning method based on CPSO-FEO, the HW/SW communication cost runs
in multicore parallel computing environment.

The paper is organized as follows. Section 2 describes the related work. The
heuristic algorithms for HW/SW partitioning are introduced in Sec. 3. The CPSO-
FEO is presented in Sec. 4. Parallel HW/SW partitioning method based on CPSO-
FEO is proposed in Sec. 5. Section 6 analyzes the performance of CPSO-FEO
experimentally. Finally, conclusions are summarized in Sec. 7.

2. Related Works

The heuristic algorithms for HW/SW partitioning problem have been widely
researched over the last decade. The traditional heuristic approaches include
hardware-oriented and software-oriented ones. The hardware-oriented approach
starts with a complete hardware solution and swaps parts to software until con-
straints are violated while the software-oriented approach means that the initial
implementation of the system is supposed to be a software solution.

The PSO is attractive for the HW/SW partitioning problem as it offers rea-
sonable coverage of the design space together with short execution time.
Ref. 30, the authors find that the PSO outperforms GA in the cost function and the execution time for solving the HW/SW partitioning problem. In Ref. 31, the authors propose a modified PSO restarting technique to avoid quick convergence, named the re-excited PSO algorithm, in order to solve the HW/SW partitioning problem. In Ref. 36, it is revealed that performance of PSO-based algorithm outperforms ILP, GA and ACO for the HW/SW partitioning problem. In Ref. 37, discrete particle swarm optimization (DPSO) and B&B algorithms are presented to solve the HW/SW partitioning problems, and DPSO is used to increase the search speed of B&B. In Ref. 38, a hybrid method of PSO and TS is proposed to solve the HW/SW partitioning problems. And TS runs for optimal solution with the result of PSO as initial input which can improve the convergence speed.

In early studies, the target architecture is supposed to consist of a single software unit and a single hardware unit. The HW/SW partitioning problem has specific optimization objectives, such as minimizing power, hardware area and communication overhead. In this paper, the HW/SW partitioning model and algorithm are not limited within an architecture and a particular objective, and the HW/SW partitioning problem is formalized as a graph partitioning problem. Therefore, HW/SW partitioning definition is general enough so that the proposed algorithm can be used in real and practical cases.

To compare and verify the proposed algorithm fairly, we discuss the partitioning problem based on the same assumptions and system model which are used in Refs. 11 and 25. In Ref. 28, the HW/SW partitioning problem is transformed into a one-dimensional one and three heuristic algorithms are proposed to determine suitable partitions to satisfy the HW/SW partitioning constraints. In Ref. 28, an efficient heuristic approach and tabu search are presented for the HW/SW partitioning that aims to minimize the hardware cost while taking into account software and communication constraints. Experimental results show that the proposed algorithms outperform the algorithm in Ref. 28 by up to 28%. In Ref. 39, NodeRank that calculates the rank of each node iteratively is proposed to solve the HW/SW partitioning problem. Experimental results show that NodeRank outperforms the algorithms in Ref. 28. It is difficult for the above heuristic algorithms to search global best solution when the scale is large.

The heuristic algorithms take too much time in addressing large-scale HW/SW partitioning problems. In previous studies, PC clusters and supercomputers were used to solve the problem. However, because of sequential computing method, the performance of such computer systems could not be easily promoted. With the rapid development of multicore CPU technique, using multicore parallel computing is an intuitive and simple way to speed up the heuristic algorithms for the HW/SW partitioning problem. To accelerate the speed of HW/SW partitioning, parallel HW/SW partitioning method based on the message passing interface (MPI) is presented in Refs. 41 and 41. GPU parallel technique is used to accelerate the HW/SW partitioning method in Ref. 42.
In this paper, to enhance the solution quality, an improved PSO which is named CPSO-FEO is proposed to solve the HW/SW partitioning problem. To accelerate HW/SW partitioning method based on CPSO-FEO, the HW/SW communication cost runs in the ordinary multicore PC platform. To the authors’ knowledge, no prior work has been done to solve the HW/SW partitioning problem using multicore parallel computing and the improved PSO.

3. Heuristic Algorithms for HW/SW Partitioning Problem

3.1. Problem definition

There is no uniform model for the HW/SW partitioning problem. The HW/SW partitioning model proposed in Ref. 11 is classical and it is widely used in modern embedded systems. To compare and verify the proposed algorithm, we discuss the partitioning problem based on the same assumptions and model which are used in Refs. 11 and 25. Given an undirected graph \( G = (V, E) \), where \( V \) is the vertex set and \( E \) is the edge set. \( V = \{v_1, v_2, \ldots, v_n\} \), where \( n \) indicates the number of nodes in the graph. \( s_i \) and \( h_i \) indicate the software and hardware costs of node \( v_i \), respectively. To minimize the hardware costs under constraint condition, the HW/SW partitioning problem can be formulated to the minimization problem \( P \) as follows:

\[
P \begin{cases}
\text{minimize} & \sum_{i=1}^{n} h_i (1 - x_i) \\
\text{subject to} & \sum_{i=1}^{n} s_i x_i + C(x) \leq R, \\
& x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n
\end{cases}
\] (1)

where \( x = (x_1, x_2, \ldots, x_n) \) indicates a solution of the HW/SW partitioning problem, \( x_i = 1 \) (\( x_i = 0 \)) denotes that the node is partitioned to software (hardware). \( C(x) \) indicates the communication cost of \( x \), and \( R \) is the constraint. The minimization problem \( P \) can be converted to the following maximization problem \( Q \):

\[
Q \begin{cases}
\text{maximize} & \sum_{i=1}^{n} h_i x_i \\
\text{subject to} & \sum_{i=1}^{n} s_i x_i + C(x) \leq R, \\
& x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n
\end{cases}
\] (2)

In Ref. 25, \( C(x) \) is replaced with \( uR \), where \( 0 \leq u \leq 1 \), then the problem \( Q \) can be converted to \( Q' \):

\[
Q' \begin{cases}
\text{maximize} & \sum_{i=1}^{n} h_i x_i \\
\text{subject to} & \sum_{i=1}^{n} s_i x_i \leq (1 - u)R, \\
& x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n
\end{cases}
\] (3)
The knapsack problem is very similar to the HW/SW partitioning problem \( Q' \). Hence, the HW/SW partitioning problem is reduced to a variation of knapsack problem and three algorithms are proposed in Ref. [25]. Alg-new3 which works best among the three algorithms is named Base in this paper. The time complexity of Base is \( O(n \log n + d(n + m)) \), where \( n \) and \( m \) indicate the number of nodes and the number of edges in the communication graph, respectively. In Ref. [28], a heuristic algorithm which is named Heur is proposed. The time complexity of Heur is \( O(n \log n + k(m + n)) \). In Ref. [29], NodeRank is presented to solve the HW/SW partitioning problem. The time complexity of NodeRank is \( O(N(n + m)\log n) \), where \( N \) is the maximum iteration time.

However, the HW/SW partitioning problem becomes far different from the standard knapsack problem when the scale becomes large. And it is difficult for the above heuristic algorithms to search for the global best solution when the large-scale HW/SW partitioning problem is treated as a variation of knapsack problem.

### 3.2. Basic particle swarm optimization

PSO, which was first introduced by Kennedy and Eberhart in 1995, emulates the social behavior of bird flocking and fish schooling[43-45]. PSO has received significant interest from researchers working in different research areas and has been applied to several real-world problems, such as robotic controllers,[46] traveling salesman problem,[47,48] feature selection,[49] and smart distribution networks.[50]

In basic PSO, xSize particles search the global best position in the \( n \)-dimensional search space. And each particle has the following attributes: a current position in the search space \( X_i \), a current velocity \( V_i \) and a personal best position \( \text{pbest}_i \) in the search space, then

\[
X_i = (X_{i1}, \ldots, X_{id}, \ldots, X_{in}),
\]

\[
V_i = (V_{i1}, \ldots, V_{id}, \ldots, V_{in}),
\]

\[
\text{pbest}_i = (\text{pbest}_{i1}, \ldots, \text{pbest}_{id}, \ldots, \text{pbest}_{in}),
\]

where \( 1 \leq i \leq \text{xSize} \) and \( 1 \leq d \leq n \). The global best position discovered by the whole population is \( \text{gbest} \).

\[
\text{gbest} = (\text{gbest}_{1}, \ldots, \text{gbest}_{d}, \ldots, \text{gbest}_{n}).
\]

And the evolution equations of PSO are

\[
V_{id}^{k+1} = wV_{id}^k + c_1r_1(\text{pbest}_{id}^k - X_{id}^k) + c_2r_2(\text{gbest}_d^k - X_{id}^k),
\]

\[
X_{id}^{k+1} = X_{id}^k + V_{id}^{k+1},
\]

where \( w \) is the inertia weight used to balance the local and global search abilities; \( c_1 \) and \( c_2 \) are learning coefficients; \( r_1 \) and \( r_2 \) are two random numbers uniformly distributed in the range \([0, 1]\); and \( k \) is the iteration number. The first
part of (8) represents the previous velocity, which provides the necessary momentum for particles to roam across the search space. The second part, known as the cognitive component, represents the personal thinking of each particle. The third part, known as the social component, represents the collaborative effect of the particles.

4. Conformity Particle Swarm Optimization with Fireworks Explosion Operation

Basic PSO has many advantages, such as simple implementation and fewer parameters. However, PSO may easily get trapped into the local optimum when solving complex multimodal optimization problems. To search the global best solution, CPSO-FEO is proposed to solve the HW/SW partitioning problem.

4.1. Conformity particle swarm optimization

It has been observed by biologists that social relationships are important for helping an animal to cope with its environment. The particles seem to have conformist mentality and tend to move towards a secure place, where there are lots of particles and there is less possibility to be attacked by natural enemy. Then the particles can keep up with the population and resist predator attack together.

Without losing the generality, we give following definitions as shown in Fig. 1.

**Definition 4.1 (Safe place).** The safe place in population is the place where there are lots of particles and there is less possibility to be attacked by natural enemy.

![Fig. 1. Secure positions (SPs) in the population.](image-url)
Definition 4.2 (Secure range). The secure range is the area within which more than $\mu$ percentage of the population is located. The secure range is denoted as SR. As shown in Fig. 1, the sheep are the particles, and the area in the red circle is SR.

Definition 4.3 (Secure position). The secure position is the center position of SR, which is denoted as $SP = (SP_1, SP_2, \ldots, SP_n)$. As shown in Fig. 1, $C_j$ is SP.

Definition 4.4 (Maximum secure distance). The maximum secure distance is the radius of SR, which is denoted as MSD. As shown in Fig. 1, the red solid line is MSD.

The $i$th particle is in SR, then

$$\|X_i - SP\| \leq MSD.$$  \hspace{1cm} (10)

The steps to compute the secure position $SP$ are as follows:

Step 1: Choose a position in the searching range randomly and set the position as SP.

Step 2: Compute the number of particles which are in SR according to (10). The number is named num.

Step 3: If $num/xSize > \mu$, then the position is the secure position. Otherwise, return to step 1.

To reduce the computational burden, the average position of the population is considered as the secure position in this paper, thus,

$$SP_d = \frac{\sum_{i=1}^{xSize} X_{id}}{xSize}.$$  \hspace{1cm} (11)

The algorithm may be trapped into the local optimum when the current global best value is almost the same as the previous one. Each particle accelerates in the direction of the secure position, then the evolution equations of CPSO are

$$V_{id}^{k+1} = wV_{id}^k + c_1r_1(pbest_{id}^k - X_{id}^k) + c_2r_2(gbest_d^k - X_{id}^k) + c_3r_3(SP_d^k - X_{id}^k),$$  \hspace{1cm} (12)

$$X_{id}^{k+1} = X_{id}^k + V_{id}^{k+1},$$  \hspace{1cm} (13)

where $c_3$ is the conformist coefficient and $r_3$ is a random number uniformly distributed in the range $[0, 1]$. The fourth part of (12) represents the conformist part, which can disturb particle distribution and make the population become various. Therefore, there is a greater possibility to increase the population diversity and avoid local optimum in CPSO.
4.2. The convergence analysis of CPSO

The convergence of CPSO is analyzed in this paper. According to (12), the velocity equation of CPSO can be converted to

$$V_{i}^{k+1} = wV_{i}^{k} + c_{1}r_{1}p_{best}^{k} + c_{2}r_{2}g_{best}^{k} + c_{3}r_{3}SP^{k} - (c_{1}r_{1} + c_{2}r_{2} + c_{3}r_{3})X_{i}^{k}.$$  

(14)

When $k \to \infty$, $V_{i}^{k}$ and $V_{i}^{k+1}$ are equal to zero. Then (14) can be converted to

$$\lim_{k \to \infty} X_{i} = \frac{c_{1}r_{1}p_{best} + c_{2}r_{2}g_{best} + c_{3}r_{3}SP}{c_{1}r_{1} + c_{2}r_{2} + c_{3}r_{3}}.$$  

(15)

As $r_{1}, r_{2}, r_{3}$ are the generated uniformly distributed random numbers in the range [0, 1], then the mean of $X_{i}$ in CPSO is

$$\lim_{k \to \infty} E(X_{i}) = E\left(\frac{c_{1}r_{1}p_{best} + c_{2}r_{2}g_{best} + c_{3}r_{3}SP}{c_{1}r_{1} + c_{2}r_{2} + c_{3}r_{3}}\right) = \frac{c_{1}p_{best} + c_{2}g_{best} + c_{3}SP}{c_{1} + c_{2} + c_{3}}.$$  

(16)

In (16), $\alpha = \frac{c_{2}}{c_{1} + c_{2} + c_{3}}$ and $\beta = \frac{c_{3}}{c_{1} + c_{2} + c_{3}}$. In the same way, the mean of $X_{i}$ in PSO is

$$\lim_{k \to \infty} E(X_{i}) = E\left(\frac{c_{1}r_{1}p_{best} + c_{2}r_{2}g_{best}}{c_{1}r_{1} + c_{2}r_{2}}\right) = \frac{c_{1}p_{best} + c_{2}g_{best}}{c_{1} + c_{2}}.$$  

(17)

In (17), $\delta = \frac{c_{2}}{c_{1} + c_{2}}$. According to (17), the diversification and convergence of PSO are excellent in earlier iterations. However, the particles approach from pbest to gbest in later iterations. Then the diversification is reduced and the algorithm is trapped into the local optimal solution. According to (16) and (17), convergence values of CPSO and PSO are different. CPSO can increase the population diversity, and avoid premature convergence and local optimal solution.

4.3. The global best position being updated by FEO

The fireworks algorithm (FWA) is a newly developed swarm intelligence algorithm based on simulating the explosion process of fireworks. Inspired by real fireworks, the main idea of FWA is to use the explosion of the fireworks to search for a feasible solution. FWA has been applied for solving many practical optimization problems. The flowchart of FWA is shown in Fig. 2.
fireworks are initialized randomly and their quality is evaluated. Subsequently, fireworks explode and generate different types of sparks within their local space. Finally, high-quality fireworks are selected among the set of candidates. The algorithm continues the search until a termination condition is reached.

The explosion operation in FWA is simple and the operation is integrated to search for a better position in CPSO-FEO, by which the global best position is updated. The initialization strategy of FEO is improved in this paper. The population of FEO is initialized by the personal best position pbest, then the numbers of explosion sparks and the explosion amplitudes of fireworks are calculated as follows:

\[ S_i = M \times \frac{f_{\text{max}} - f(p\text{best}_i) + \varepsilon}{\sum_{t=1}^{\text{Size}} (f_{\text{max}} - f(p\text{best}_t)) + \varepsilon}, \]

\[ A_i = \hat{A} \times \frac{f(p\text{best}_i) - f_{\text{min}} + \varepsilon}{\sum_{t=1}^{\text{Size}} (f(p\text{best}_t) - f_{\text{min}}) + \varepsilon}, \]

where \( M \) is a parameter controlling the overall spark numbers and \( \hat{A} \) is a parameter controlling the overall explosion amplitude. \( f_{\text{max}} \) stands for the firework with the worst fitness, \( f_{\text{min}} \) stands for the firework with the best fitness and \( \varepsilon \) is the machine epsilon.

After the fireworks explosion operation, the position of the \( i \)th particle’s best position is as follows:

\[ p\text{best}_{i} = p\text{best}_i \times r_4 \times A_i, \]
where \( r_4 \) is a random number uniformly distributed in the range \([0, 1]\). The FEO is used to search for a better position in each iteration of CPSO-FEO. If the best searching position is better than the current global best position, then the current global best position is replaced by the searching one. FEO is the noniterative algorithm and has low overhead execution time. The searching density and intensification can be increased to update the global best position using FEO. Then the search accuracy and solution quality can be enhanced.

5. Parallel HW/SW Partitioning Method Based on CPSO-FEO

5.1. HW/SW partitioning method based on CPSO-FEO

As the HW/SW partitioning is treated as a discrete optimization problem, the discrete CPSO-FEO is adopted in this paper. In discrete CPSO-FEO, the position of particle in each iteration is defined as follows:

\[
X_{id}^k = \begin{cases} 
1 & \text{if } r_5 < \text{sig}(V_{id}^k), \\
0 & \text{otherwise,}
\end{cases}
\]  

(21)

where the sigmoid function \( \text{sig}(V_{id}^k) = \frac{1}{1 + e^{-V_{id}^k}} \) and \( r_5 \) is a random number uniformly distributed in the range \([0, 1]\). If \( r_5 \) is smaller than \( \text{sig}(V_{id}^k) \), then \( X_{id}^k \) is 1. Otherwise \( X_{id}^k \) is 0. The position of particle is the HW/SW partitioning solution and the fitness of particle is the hardware cost. The discrete CPSO-FEO is used to optimize the minimization problem \( P \) in (1).

As the performance of NodeRank is high among the heuristic algorithms, the particles are initialized by the algorithm NodeRank. Then the HW/SW partitioning method based on CPSO-FEO (Algorithm 5.1) is as follows.

In CPSO-FEO, \( X \) is initialized by the partition solution obtained by NodeRank in line 2. \( xSize \) is the number of particles and \( n \) is the number of nodes. In line 3, \( fPbest \) is the fitness of pbest and \( fGbest \) is the fitness of gbest. In lines 18–22, the algorithm may be trapped into the local optimum when the previous and current values of \( fGbest \) are almost the same. Then the velocity and the position of each particle are updated by (12) and (13), respectively. Otherwise, they are updated by (8) and (9), respectively. Adjust\_In is used to search for a better solution in line 23. The global best particle is updated by the FEO in line 24, and the algorithm FEO (Algorithm 5.2) is as follows.

To describe the procedure of CPSO-FEO more clearly, the flowchart of CPSO-FEO is shown in Fig. 3.

In CPSO-FEO, the calculation of the HW/SW communication cost \( C(x) \) in line 10 is as follows:

\[
C(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}|x_i - x_j|,
\]  

(22)
Algorithm 5.1. CPSO-FEO

Input: Communication graph \(G\) and the constraint \(R\).
Output: The global best partition solution \(gbest\) and its hardware cost \(fgbest\).

1. \(X_{nr} := \) the partition solution obtained by NodeRank;
2. \(X = (X_{nr}, X_{nr}, \ldots, X_{nr})\), \(V = \text{rand}(x\text{Size}, n)\); /*\(X\) and \(V\) are matrixes with \(x\text{Size}\) columns and \(n\) rows. */
3. \(pbest = (0, 0, \ldots, 0), fpgbest = (0, 0, \ldots, 0), gbest = (0, 0, \ldots, 0), fgbest = 0, fgbestold = 1, \text{iter} = 0;\)
4. Initialize other parameters, such as \(w, c_1, c_2, c_3\) and \(\text{epsx}\), the individual number \(x\text{Size}\) and the maximum iteration time \(\text{iter}_{\text{num}};\)
5. repeat
6. \(\text{iter} := \text{iter} + 1;\)
7. \(X\) is discretized by (21);
8. \(sx = S(X), hx = H(X); /*\text{Compute the software cost and hardware cost.} */\)
9. for \(i = 1 \text{ to } x\text{Size}\) do
10. \(\text{if } sx(i) + C(X_i) > R \text{ then }\)
11. \(X_i = X_{nr}, hx(i) = H(X_i);\)
12. end if
13. end for
14. Update \(pbest, gbest, fpgbest\) and \(fgbest;\)
15. for \(j = 1 \text{ to } n\) do
16. The secure point \(SP\) is computed according to (11);
17. end for
18. if \(|fgbestold - fgbest| > \text{epsx}\) then
19. The velocity and the position of particle are updated by (8) and (9), respectively;
20. else
21. The velocity and the position of particle are updated by (12) and (13), respectively;
22. end if /*\text{epsx is a small floating-point number.} */
23. \(gbest = \text{Adjust}_\text{In}(gbest), fgbest = H(gbest);\)
24. The global best particle is updated by FEO;
25. \(fgbestold = fgbest;\)
26. until (\(\text{iter} > \text{iter}_{\text{num}}\))

where \(c_{ij}\) indicates the communication cost between \(v_i\) and \(v_j\) if they are in different contexts.

**Theorem 5.1.** The time complexity of the HW/SW communication cost \(C(x)\) is \(O(n^2)\), where \(n\) indicates the number of nodes.

**Proof.** In (22), there are two layers of iteration. And each iteration can be implemented in \(O(n)\). Therefore, the time complexity of \(C(x)\) is \(O(n^2)\). \(\square\)
Algorithm 5.2. FEO

**Input:** The personal best position $p_{best}$.

**Output:** The updated global best position $g_{best}$.

1: Initialize the parameters, such as $\hat{A}$ and $M$;
2: After the fireworks explosion operation, compute position of $p_{best}$ according to (20);
3: Computer the position $p_{best}'$ which is the best position among $p_{best}'$;
4: if $p_{best}'$ is a feasible solution of $P$ and $p_{best}'$ is better than $g_{best}$ then
5: $g_{best}$ is replaced by $p_{best}'$;
6: end if

Fig. 3. Flowchart of CPSO-FEO.

**Theorem 5.2.** The time complexity of CPSO-FEO is $O(\text{iter\_num} \times \text{xSize} \times n^2)$, where $n$ is the number of nodes, $m$ is the number of edges, $\text{xSize}$ is the number of particles in the population and $\text{iter\_num}$ is the maximum iteration time of CPSO-FEO.
LSSTCS: An Efficient PSO for Large-Scale HW/SW Co-Design System

Algorithm 5.3. PSO

Input: Communication graph \( G \) and the constraint \( R \).
Output: The global best partition solution \( gbest \) and its hardware cost \( fgbest \).

1. \( X_{nr} := \) the partition solution obtained by NodeRank;
2. \( X = (X_{nr}, X_{nr}, \ldots, X_{nr}) \) \( V = \text{rand}(xSize, n) \); /* \( X \) and \( V \) are matrixes with \( xSize \) columns and \( n \) rows.*/
3. \( \text{pbest} = (0, 0, \ldots, 0), \text{fpbest} = (0, 0, \ldots, 0), \text{gbest} = (0, 0, \ldots, 0), \text{fgbest} = 0, \text{fgbestold} = 1, \text{iter} = 0; \)
4. Initialize other parameters, such as \( w, c_1 \) and \( c_2 \), the individual number \( xSize \) and the maximum iteration time \( \text{iter}_{num} \);
5. repeat
6. \( \text{iter} := \text{iter} + 1; \)
7. \( X \) is discretized by (21);
8. \( \text{sx} = S(X), \text{hx} = H(X); /* \text{Compute the software and hardware costs.} */ \)
9. for \( i = 1 \) to \( xSize \) do
10. if \( \text{sx}(i) + C(X_i) > R \) then
11. \( X_i = X_{nr}, \text{hx}(i) = H(X_i); \)
12. end if
13. end for
14. Update \( \text{pbest}, \text{gbest}, \text{fpbest} \) and \( \text{fgbest} \);
15. The velocity and the position of particle are updated by (8) and (9), respectively;
16. until (\( \text{iter} > \text{iter}_{num} \))

Proof. In CPSO-FEO, the time complexity of NodeRank is \( O(N(n + m) \log n) \) in line 1. In lines \( 9-13 \), the time complexity is \( O(xSize \times n^2) \). It takes \( O(xSize \times n) \) to update \( \text{pbest}, \text{gbest}, \text{fpbest} \) and \( \text{fgbest} \) in line 14. Adjust in line 23 can be implemented in \( O(\log n(n + m)) \) time. The time complexity of FEO is \( O(xSize \times M) \), and we set \( M < n \). Thus the time complexity of FEO is lower than \( O(xSize \times n) \). Compared to \( \log n(n + m) \), \( xSize \times n^2 \) is larger. The maximum iteration of CPSO-FEO is \( \text{iter}_{num} \). Therefore, the time complexity of CPSO-FEO is \( O(\text{iter}_{num} \times xSize \times n^2) \).

HW/SW partitioning method based on basic PSO is named PSO (Algorithm 5.3) in this paper. The computing steps of PSO and CPSO-FEO are almost the same. In PSO, the velocity and the position of particle are updated by (8) and (9). The global best position is not updated by FEO in PSO.

Theorem 5.3. The time complexity of CPSO-FEO is the same as that of PSO.

Proof. In CPSO-FEO, the time complexity of updating the velocity and position in line 21 is the same as that in line 19. The FEO can be implemented in \( O(xSize \times n) \).
time. Then the time complexity of updating the global best particle by FEO is lower than that of computing the HW/SW communication cost in lines 9–13. Therefore, the time complexity of CPSO-FEO is the same as that of PSO.

**Theorem 5.4.** The space complexity of CPSO-FEO is $O(\text{xSize} \times n)$, which is the same as that of PSO, where $n$ is the number of nodes and xSize is the number of particles.

**Proof.** In CPSO-FEO, each particle approaches the global best position, each particle’s individual best position and the secure position. The space complexity of the global best position and the secure position is $O(n)$. The space complexity of each particle’s individual best position is $O(\text{xSize} \times n)$. Therefore, the space complexity of CPSO-FEO is $O(\text{xSize} \times n)$. In PSO, each particle approaches the global best position and each particle’s individual best position. Therefore, the space complexity of PSO is $O(\text{xSize} \times n)$, which is the same as that of CPSO-FEO.

### 5.2. Parallel HW/SW partitioning method based on CPSO-FEO

Parallel computing aims to solve a problem faster by dividing the problem into a set of subproblems which are simultaneously executed by multiple independent computing nodes. In CPSO-FEO, the HW/SW communication cost is considered in each iteration. As the time complexity of the HW/SW communication cost $C(x)$ is $O(n^2)$, the calculation of $C(x)$ is excessive. In order to take full advantage of the multicore processor’s computing resources and accelerate the speed of CPSO-FEO, the HW/SW communication cost computing which is the most time-consuming process for HW/SW partitioning method runs in the ordinary multicore PC platform.

The commonly used parallelism methods consist of task parallelism, pipeline parallelism and data parallelism. The task parallelism is adopted in this paper. The multicore parallel task is computing the HW/SW communication cost $C(x)$ in this paper. And the computing nodes are the cores in the processor. Thus the mechanism of the multicore parallel computing in CPSO-FEO is shown in Fig. 4.

As shown in Fig. 4, task parallelism distributes tasks across different parallel computing nodes. Tasks are implemented as threads, and each thread is executed in a different core to process the same or different data simultaneously. All the computing threads in different nodes are joined, and the HW/SW communication cost computing result is obtained.

In CPSO-FEO, the communication cost computing for xSize particles is in lines 9–12. In NodeRank, the communication cost computing is in line 18. And the two processes run in the multicore parallel computing environment. Therefore, the runtime of CPSO-FEO can be reduced efficiently.
6. Experimental Results and Analysis

In order to evaluate the performance of CPSO-FEO, the algorithm is applied to solve several acknowledged benchmarks of the HW/SW partitioning problems. The experiment environment is as follows: CPU: i7-4770 @ 3.4 GHz; physical memory: 16 GB; and software: Matlab R2014b. The computer has eight cores.

We adopt the same experimental parameters in Ref. [25]. Software costs $s_i$ are generated as uniform random numbers from the interval $[1, 100]$. And hardware costs $h_i$ are generated as random numbers from a normal distribution with expected value $k \times s_i$ and a given standard deviation. The value of $k$ only corresponds to the choice of units for software and hardware costs. The communication costs are generated as uniform random numbers from the interval $[0, 2\rho s_{\text{max}}]$, where $s_{\text{max}}$ is the highest software cost. $\rho$ is named the communication-to-computation ratio (CCR), and it is taken as 0.1, 1 and 10 corresponding to computation-intensive case, intermediate case and communication-intensive case, respectively. The constraint $R$ is randomly generated as a uniform random number from the interval $[0, 0.5\sum s_i]$ or $[0.5\sum s_i, \sum s_i]$. The two cases are indicated as $R_{\text{low}}$ and $R_{\text{high}}$, respectively. There are six cases with different values of CCR and $R$, which are shown in Table 1.

<table>
<thead>
<tr>
<th>Case</th>
<th>CCR</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>Low</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Low</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>High</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>Low</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>High</td>
</tr>
</tbody>
</table>
The characteristics of the test benchmarks are summarized in Table 2, in which \( n \) and \( m \) indicate the number of nodes and the number of edges in the communication graph, respectively.

The algorithms Base, Heur, NodeRank, PSO and CPSO-FEO are compared to solve the acknowledged benchmarks in Table 2. In CPSO-FEO and PSO, inertia factor \((w)\) is decreasing from 0.9 to 0.7 linearly with the iterations, the individual number \( x\text{Size} \) is 40 and the maximum iteration number is 50. As the important parameters, the learning coefficients \( c_1 \) and \( c_2 \) and the conformist coefficient \( c_3 \) have great effect on the performance of CPSO-FEO. In this paper, the appropriate values of \( c_1 \), \( c_2 \) and \( c_3 \) are obtained by experiments. In CPSO-FEO, \( c_1 = 1.5 \), \( c_2 = 1.5 \), \( c_3 = 1 \), \( \epsilon_{psx} = 1 \), \( \lambda = 0.5 \) and \( M = 20 \). In PSO \( c_1 = 2 \) and \( c_2 = 2 \). The parameters of NodeRank are set as the same values in Ref. 28. Without loss of generality, we choose 100 as the number of random instances for statistical comparison in our empirical study. The solution quality and runtime of algorithms are compared.

### 6.1. Solution quality

In this subsection, the solution quality of CPSO-FEO is evaluated against other popular HW/SW partitioning algorithms to show how much the quality can be improved by CPSO-FEO.

In order to compare the solution quality of algorithms, we define improvement of algorithm A over algorithm B as

\[
\text{imp} = \left( 1 - \frac{\text{hardware cost of A}}{\text{hardware cost of B}} \right) \times 100\%.
\]  

In (23), \( \text{imp} > 0 \), \( \text{imp} = 0 \) and \( \text{imp} < 0 \) reflect that the performance of algorithm A is better than, the same as and worse than algorithm B, respectively.

Figure 5 shows the improvements of different algorithms over Base, averaged over 100 instances on different cases. Abscissa represents the problem size while the vertical axis indicates the improvement of different algorithms over Base.
Fig. 5. Improvements of different algorithms over Base, averaged over 100 instances on different cases, namely (a) case 1, (b) case 2, (c) case 3, (d) case 4, (e) case 5 and (f) case 6.
In Fig. 5, generally, the solutions found by CPSO-FEO are better than or similar to the solutions found by the other algorithms. The solutions found by NodeRank, PSO and CPSO-FEO are almost the same when the communication cost is small, e.g. as shown in Fig. 5(a). This is because the HW/SW partitioning problems with small communication cost do not have a significant impact on the constraint. The optimal solutions can be easily found by other algorithms. On the other hand, with the increase of the communication cost, the constraints of the partitioning problem become far different from those of the standard knapsack problem. Therefore, it becomes harder to search for the optimal solution. Since the global search ability of CPSO-FEO is stronger, the improvement differences among CPSO-FEO, PSO and NodeRank become greater, e.g. as shown in Fig. 5(f).

In order to explore the worst-case performance of CPSO-FEO, we investigate the distribution of improvement over the algorithm Base for 100 random instances. Without loss of generality, the imp values are collected from $-50\%$ to $50\%$. This corresponds to the distribution interval $[-50, -45, \ldots, -10, -5, 0, 5, 10, \ldots, 45, 50]$ with unit length of 5 in $x$-axis. We choose the representative benchmark Random2. And the distribution of improvements over Base on different cases is shown in Fig. 6.

From Fig. 6, the corresponding improvements are distributed in a relatively large interval for the cases of small communication cost, e.g. as shown in Fig. 6(a). And the interval becomes smaller with the increasing communication cost, e.g. as shown in Fig. 6(f). In the maximum distributed interval, the count of CPSO-FEO is the largest one. Thus, CPSO-FEO can produce higher quality solutions than the other algorithms for most instances especially when the communication cost is relatively large. Since the population of CPSO-FEO is initialized by NodeRank, the worst-case performance of CPSO-FEO is better than that of NodeRank.

### 6.2. Runtimes of algorithms

It usually takes too much time in addressing large-scale HW/SW partitioning problems for heuristic algorithms. In this subsection, the runtime of CPSO-FEO is evaluated against other popular HW/SW partitioning algorithms to show how much the runtime can be reduced by CPSO-FEO.

Runtime of HW/SW partitioning algorithms is used to evaluate the multicore parallel computing efficiency. The communication cost computing of CPSO-FEO runs in the multicore parallel computing environment, while the computing of other algorithms runs in sequential computing environment. Figure 7 shows the average runtimes of different algorithms on different cases. Abscissa represents the problem number while the vertical axis indicates the runtimes of different algorithms.

As shown in Fig. 7 the runtime of CPSO-FEO is longer than that of PSO when the size is small. This is because there are extra operations which take extra runtime in multicore parallel computing, such as thread creation, task creation and data communication. Generally, the advantage of parallel CPSO-FEO in runtime becomes more obvious as the problem size increases. In order to compare...
LSSTCS: An Efficient PSO for Large-Scale HW/SW Co-Design System

Fig. 6. Distribution of improvements over Base, 100 random instances on different cases with size = 8,000, where (a) case 1, (b) case 2, (c) case 3, (d) case 4, (e) case 5 and (f) case 6.
Fig. 7. Runtimes of different algorithms, averaged over 100 random instances on different cases, where (a) case 1, (b) case 2, (c) case 3, (d) case 4, (e) case 5 and (f) case 6.
the runtimes of the algorithms more clearly, Tables 3 and 4 show the runtimes of different algorithms on cases 1 and 6, respectively.

As shown in Tables 3 and 4, the runtimes of the algorithms become longer when the problem size becomes larger. Generally, the runtimes of the algorithms in Table 4 are longer than the corresponding values in Table 3. This is because the computational burden becomes greater when the HW/SW communication cost becomes larger. As the algorithms PSO and CPSO-FEO are initialized by NodeRank, the runtimes of PSO and CPSO-FEO are longer than that of NodeRank. The runtime of CPSO-FEO is longer than that of PSO when the size is smaller than 16,000. This is because the computational burden of CPSO-FEO is larger than that of PSO and the advantage of parallel computing is not obvious when the size is small. On the other hand, the HW/SW communication cost becomes excessive when the size is large. The advantage of the HW/SW communication cost computing in multicore PC platform is obvious. And the runtime of CPSO-FEO is shorter than that of PSO when the size is equal to or greater than 16,000.

Table 3. Comparison of runtimes (in seconds) on case 1, averaged over 100 instances.

<table>
<thead>
<tr>
<th>Number</th>
<th>Base</th>
<th>Heur</th>
<th>NodeRank</th>
<th>PSO</th>
<th>CPSO-FEO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0022</td>
<td>0.0017</td>
<td>0.0483</td>
<td>0.0522</td>
<td>0.6441</td>
</tr>
<tr>
<td>2</td>
<td>0.0023</td>
<td>0.0023</td>
<td>0.0542</td>
<td>0.0601</td>
<td>0.6783</td>
</tr>
<tr>
<td>3</td>
<td>0.0024</td>
<td>0.0024</td>
<td>0.0425</td>
<td>0.1065</td>
<td>0.9044</td>
</tr>
<tr>
<td>4</td>
<td>0.0028</td>
<td>0.0029</td>
<td>0.0655</td>
<td>0.1636</td>
<td>0.8184</td>
</tr>
<tr>
<td>5</td>
<td>0.0041</td>
<td>0.0073</td>
<td>0.1007</td>
<td>0.2995</td>
<td>1.3327</td>
</tr>
<tr>
<td>6</td>
<td>0.0095</td>
<td>0.0350</td>
<td>0.4705</td>
<td>1.1135</td>
<td>2.0074</td>
</tr>
<tr>
<td>7</td>
<td>0.0110</td>
<td>0.0904</td>
<td>1.0564</td>
<td>2.6011</td>
<td>3.6649</td>
</tr>
<tr>
<td>8</td>
<td>0.0011</td>
<td>0.0515</td>
<td>0.9114</td>
<td>2.2613</td>
<td>2.9994</td>
</tr>
<tr>
<td>9</td>
<td>0.0167</td>
<td>0.1514</td>
<td>1.7535</td>
<td>4.0268</td>
<td>4.9204</td>
</tr>
<tr>
<td>10</td>
<td>0.0164</td>
<td>0.1147</td>
<td>1.3408</td>
<td>3.1830</td>
<td>3.6906</td>
</tr>
<tr>
<td>11</td>
<td>0.0172</td>
<td>0.1930</td>
<td>1.7163</td>
<td>5.0121</td>
<td>5.4538</td>
</tr>
<tr>
<td>12</td>
<td>0.0221</td>
<td>0.3112</td>
<td>3.5491</td>
<td>8.1575</td>
<td>7.9035</td>
</tr>
<tr>
<td>13</td>
<td>0.0228</td>
<td>0.4242</td>
<td>4.6878</td>
<td>10.6033</td>
<td>9.6864</td>
</tr>
<tr>
<td>14</td>
<td>0.0246</td>
<td>0.5103</td>
<td>5.8356</td>
<td>13.5581</td>
<td>12.0697</td>
</tr>
</tbody>
</table>

Table 4. Comparison of runtimes (in seconds) on case 6, averaged over 100 instances.

<table>
<thead>
<tr>
<th>Number</th>
<th>Base</th>
<th>Heur</th>
<th>NodeRank</th>
<th>PSO</th>
<th>CPSO-FEO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0029</td>
<td>0.0017</td>
<td>0.0182</td>
<td>0.0398</td>
<td>0.8427</td>
</tr>
<tr>
<td>2</td>
<td>0.0036</td>
<td>0.0015</td>
<td>0.0321</td>
<td>0.0529</td>
<td>0.8806</td>
</tr>
<tr>
<td>3</td>
<td>0.0036</td>
<td>0.0042</td>
<td>0.0479</td>
<td>0.0647</td>
<td>0.9220</td>
</tr>
<tr>
<td>4</td>
<td>0.0040</td>
<td>0.0036</td>
<td>0.0756</td>
<td>0.1864</td>
<td>1.1640</td>
</tr>
<tr>
<td>5</td>
<td>0.0044</td>
<td>0.0094</td>
<td>0.1156</td>
<td>0.2983</td>
<td>1.1782</td>
</tr>
<tr>
<td>6</td>
<td>0.0128</td>
<td>0.0501</td>
<td>0.5332</td>
<td>1.3760</td>
<td>2.1740</td>
</tr>
<tr>
<td>7</td>
<td>0.0141</td>
<td>0.1098</td>
<td>1.1155</td>
<td>2.6475</td>
<td>3.3644</td>
</tr>
<tr>
<td>8</td>
<td>0.0148</td>
<td>0.1019</td>
<td>1.0038</td>
<td>2.5487</td>
<td>3.2080</td>
</tr>
<tr>
<td>9</td>
<td>0.0258</td>
<td>0.1949</td>
<td>1.9974</td>
<td>4.5625</td>
<td>5.1648</td>
</tr>
<tr>
<td>10</td>
<td>0.0259</td>
<td>0.1546</td>
<td>1.4715</td>
<td>3.7564</td>
<td>4.2818</td>
</tr>
<tr>
<td>11</td>
<td>0.0264</td>
<td>0.2336</td>
<td>2.2486</td>
<td>5.5025</td>
<td>5.7542</td>
</tr>
<tr>
<td>12</td>
<td>0.0302</td>
<td>0.4188</td>
<td>4.0628</td>
<td>9.5064</td>
<td>8.7979</td>
</tr>
<tr>
<td>13</td>
<td>0.0306</td>
<td>0.3084</td>
<td>4.5112</td>
<td>12.5340</td>
<td>11.0719</td>
</tr>
<tr>
<td>14</td>
<td>0.0327</td>
<td>0.6199</td>
<td>5.9998</td>
<td>14.2186</td>
<td>12.1379</td>
</tr>
</tbody>
</table>
Table 5. Large-scale HW/SW partitioning problems.

<table>
<thead>
<tr>
<th>Name</th>
<th>n</th>
<th>m</th>
<th>Size(2n+3m)</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random10</td>
<td>3,000</td>
<td>3,000</td>
<td>15,000</td>
<td>15</td>
</tr>
<tr>
<td>Random11</td>
<td>4,000</td>
<td>4,000</td>
<td>20,000</td>
<td>16</td>
</tr>
<tr>
<td>Random12</td>
<td>5,000</td>
<td>5,000</td>
<td>25,000</td>
<td>17</td>
</tr>
<tr>
<td>Random13</td>
<td>6,000</td>
<td>6,000</td>
<td>30,000</td>
<td>18</td>
</tr>
<tr>
<td>Random14</td>
<td>7,000</td>
<td>7,000</td>
<td>35,000</td>
<td>19</td>
</tr>
<tr>
<td>Random15</td>
<td>7,000</td>
<td>14,000</td>
<td>56,000</td>
<td>20</td>
</tr>
<tr>
<td>Random16</td>
<td>8,000</td>
<td>8,000</td>
<td>40,000</td>
<td>21</td>
</tr>
<tr>
<td>Random17</td>
<td>8,000</td>
<td>16,000</td>
<td>64,000</td>
<td>22</td>
</tr>
<tr>
<td>Random18</td>
<td>9,000</td>
<td>9,000</td>
<td>45,000</td>
<td>23</td>
</tr>
<tr>
<td>Random19</td>
<td>9,000</td>
<td>18,000</td>
<td>72,000</td>
<td>24</td>
</tr>
<tr>
<td>Random20</td>
<td>10,000</td>
<td>10,000</td>
<td>50,000</td>
<td>25</td>
</tr>
<tr>
<td>Random21</td>
<td>10,000</td>
<td>20,000</td>
<td>80,000</td>
<td>26</td>
</tr>
</tbody>
</table>

6.3. The large-scale HW/SW partitioning

Because of the development of the design complexity in modern embedded systems, the scale of HW/SW partitioning becomes larger. In order to analyze the performance of CPSO-FEO further, the algorithm is used to solve large-scale HW/SW partitioning problems which are shown in Table 5.

The algorithms Base, Heur, NodeRank, PSO and CPSO-FEO are compared to solve the large-scale HW/SW partitioning problem in the typical case viz. case 3. Figures 8 and 9 show the improvements and runtimes of different algorithms over Base, averaged over 100 instances on cases 3, respectively.

As shown in Figs. 8 and 9, the difference between PSO and NodeRank becomes smaller and the difference between CPSO-FEO and PSO becomes larger. This is because PSO gets trapped into local optimum and CPSO-FEO has stronger search
ability for large-scale HW/SW partitioning. And the solution quality of CPSO-FEO is enhanced distinctly.

In order to compare the runtimes of the algorithms for large-scale HW/SW partitioning problems more clearly, Table 6 shows the runtimes of different algorithms on case 3.

As shown in Table 6, the runtime of CPSO-FEO is shorter than that of PSO by 65.8046s when the size is 80,000. The multicore parallel computing is effective. Therefore, the parallel CPSO-FEO is an efficient method to solve the large-scale HW/SW partitioning problem.

7. Conclusions

A parallel HW/SW partitioning method based on CPSO-FEO is proposed. In CPSO-FEO, the particles move towards the secure point, and the searching
population remains varied. Then CPSO fights premature convergence and avoids local optimum. To improve the initialization strategy, the FEO is integrated to update the global best position in the searching population. The search accuracy and the solution quality are enhanced. The HW/SW communication cost runs in the ordinary multicore PC platform, and the runtime is reduced efficiently in the experiments.

In future, we will continue our work as follows but not limited. Firstly, we will absorb other strategies from the heuristic methods\textsuperscript{63–66} to improve the performance of CPSO-FEO. Secondly, the GPUs are available on common PC platforms with high performance and low cost. We will accelerate CPSO-FEO with parallel computing method on the GPU platforms\textsuperscript{67,68}. Thirdly, we will extend the proposed algorithm to other areas, such as video tracking\textsuperscript{69–71}, images processing\textsuperscript{72–75}, CAD\textsuperscript{76–78}, cooperative information system\textsuperscript{4–8}, CSCW\textsuperscript{79–81} and social computing\textsuperscript{82–84}.

Acknowledgments

The work was supported by the National Natural Science Foundation of China Under the Grant No. 61472289 and the National Key Research and Development Project Under the Grant No. 2016YFC0106305.

References

X. Yan et al.


75. Y. L. Chen, F. Z. He, Y. Q. Wu and N. Hou, A local start search algorithm to compute exact Hausdorff distance for arbitrary point sets, Pattern Recognit. 67 (2017) 139–148.


