Soft Comput
DOI 10.1007/s00500-016-2125-y

METHODOLOGIES AND APPLICATION

Universal partially evolved parallelization of MOEA/D for multi-objective optimization on message-passing clusters

Weiqin Ying[1] · **Yuehong Xie**[1] · **Yu Wu**[2] · **Bingshen Wu**[1] · **Shiyun Chen**[1] · **Weipeng He**[1]

Abstract This paper presents a universal partially evolved parallelization of the multi-objective evolutionary algorithm based on decomposition (MOEA/D) for multi-objective optimization on message-passing clusters to reduce its computation time. The partially evolved MOEA/D (peMOEA/D) is suitable not only for the bi-objective space, but also for higher dimensional objective spaces by using a partially evolved island model. This model improves the algorithm universality and population diversity by keeping a subpopulation equal in size to the entire population, but only evolving a partial (equal to a partition size) subpopulation on each separate processor in a cluster. Furthermore, a nearest-neighbor partitioning approach, hybrid migration policy and adaptive neighbor-based topology are adopted in the peMOEA/D. The fat partitions generated by the nearest-neighbor partitioning can reduce migration traffic of elitist individuals across subpopulations. Then, hybrid migration of both elitist individuals and utopian points helps separate subpopulations to cooperate in guiding the search quickly towards a complete front. Next, the adaptive neighbor-based topology connects neighbor partitions only and achieves an excellent balance between convergence speed and migration traffic. Experimental results on benchmark multi-objective optimization problems with two or more objectives demonstrate the satisfactory overall performance of the peMOEA/D in terms of both convergence performance and speedup on message-passing clusters.

Keywords Evolutionary algorithm · Multi-objective optimization · Parallelization · Decomposition · Message-passing clusters

1 Introduction

Many real-world optimization problems involve simultaneous optimization of multiple, possibly conflicting objectives (Zhou et al. 2011). Such multi-objective optimization problems (MOPs) generally require reconciliation of several conflicting objectives. However, solving MOPs using traditional deterministic methods is hard. Evolutionary algorithms are suitable and have been widely used for solving MOPs since they can simultaneously achieve a set of trade-off solutions in a single run. Many multi-objective optimization evolutionary algorithms (MOEAs) have been developed over the last twenty years. Most of these MOEAs, including the improved non-dominated sorting genetic algorithm, NSGA-II (Deb et al. 2002a) and the improved strength Pareto evolutionary algorithm (Zitzler et al. 2001), are based on the concept of Pareto dominance. In recent years, various novel decomposition-based MOEAs, such as the multi-objective evolutionary algorithm based on decomposition (MOEA/D) (Zhang and Li 2007) and the conical area evolutionary algorithm (CAEA) (Ying et al. 2012, 2015), have become popular and been widely used for MOPs. A MOEA/D explicitly decomposes a MOP into a certain number of scalar optimization subproblems by borrowing the idea of decomposition from traditional methods so that the offspring for each subproblem need only update the individuals for its neighbor subproblems. By using decomposition, MOEA/Ds gener-

Communicated by V. Loia.

✉ Weiqin Ying
yingweiqin@scut.edu.cn

[1] School of Software Engineering, South China University of Technology, Guangzhou 510006, China

[2] School of Computer Science and Educational Software, Guangzhou University, Guangzhou 510006, China

Published online: 28 March 2016

ally exhibit superior overall performance compared with dominance-based MOEAs.

Nevertheless, one of the potential disadvantages of MOEAs is that they could consume much computation time before generating satisfactory solutions. This is particularly true for MOPs with complicated and expensive evaluation procedures, like protein structure prediction problems (Cutello et al. 2006), truss structure design problems (Coello and Christiansen 2000), wireless sensor network layout problems (Molina et al. 2008) and so on. Thus, parallel and distributed computing needs to be considered as a mechanism for reducing the computation time of MOEAs when solving complicated problems. Popular parallel MOEAs can generally be classified into two broad categories: master–slave models and coarse-grained models. Master–slave parallel models distribute the calculation of objective functions only over a certain number of processing units to parallelize the calculation of objective functions (van Veldhuizen et al. 2003; Streichert et al. 2005; Zaharie et al. 2008). Coarse-grained parallel models, also called island models, divide the large-scale population into multiple small-scale subpopulations evolved by multiple local MOEAs (Deb et al. 2003; Branke et al. 2004; Zaharie et al. 2008; Cheshmehgaz et al. 2013). Although both categories of parallel MOEAs can, to a certain extent, reduce the computation time of MOEAs, coarse-grained models seem more suited to MOEA/Ds when considering the decomposition idea used by these.

To reduce the computation time of a MOEA/D, several shared-memory parallel versions (Nebro and Durillo 2010; Durillo et al. 2011) of the MOEA/D have been designed for shared-memory multi-processors. While the population is divided into a certain number of coarse-grained subpopulations logically and each thread evolves a certain subpopulation in any of these shared-memory parallel MOEA/Ds, all the subpopulations are stored and maintained physically within the same shared memory hardware. Consequently, every thread can directly access neighbor individuals beyond its own subpopulation during the update operation of neighbor individuals in the MOEA/D. Additionally, all subpopulations in a shared-memory MOEA/D maintain only one common utopian point in the same shared memory hardware, which every thread can share and access. As a result, these shared-memory parallel MOEA/Ds are suitable only for shared-memory multi-processors and are not yet available for message-passing clusters with distributed-memory.

In this paper, we propose a universal parallel partially evolved MOEA/D (peMOEA/D) for multi-objective optimization on message-passing clusters with distributed-memory. In this study, we first designed a parallel overlapped partitioning MOEA/D (opMOEA/D) to transplant the shared-memory MOEA/Ds to message-passing clusters by appending overlapping individuals on separate processors. However, the opMOEA/D is only suitable for bi-objective optimization. Hence, a key partially evolved island model is introduced in the peMOEA/D to extend the parallelization of MOEA/D on clusters from the bi-objective space to three or higher dimensional objective spaces. Furthermore, a nearest-neighbor partitioning approach, hybrid migration policy and adaptive neighbor-based topology are adopted in the peMOEA/D to improve convergence performance and speedup.

The remainder of this paper is organized as follows: Section 2 gives a brief introduction of the overlapped partitioning parallelization of the MOEA/D for bi-objective optimization. The universal partially evolved parallelization of the MOEA/D is presented in detail for multi-objective optimization in Sect. 3. Experimental results and analysis on benchmark MOPs with two and more objectives are presented to validate the effectiveness of the universal partially evolved MOEA/D in Sect. 4. Finally, our conclusions are given in Sect. 5.

2 Overlapped partitioning parallelization of the MOEA/D for bi-objective optimization

In this section, we first briefly introduce the parallel overlapped partitioning MOEA/D for bi-objective optimization, which can be executed on clusters using the message passing interface (MPI) (Clarke et al. 1994) by extending the basic idea of the shared-memory parallel MOEA/D executed on shared-memory multiprocessors. In the opMOEA/D, a subpopulation consists of a non-overlapping partition and some extra overlapping individuals for updating neighbors within a separate processor according to an overlapped partitioning island model, while an elitist policy and linear topology for migration are adopted to share elitist individuals across separate processors.

2.1 Overlapped partitioning island model

In the MOEA/D, a MOP is decomposed into a series of subproblems, each of which is assigned one of the uniformly distributed weight vectors in Definition 1, which aggregates all individual objectives into a single scalar objective. Then, every individual in the population of the MOEA/D is in charge of optimizing a different subproblem. In essence, every individual is assigned one particular weight vector in the MOEA/D, called a weighted individual.

Definition 1 (*Uniform spread of weight vectors*) Let H be the number of divisions for every dimension and $\mathbf{h} = (h_0, h_1, \ldots, h_{m-1})$ be a m-dimensional vector composed of integers $h_i \in [0..H]$, $i = 0, 1, \ldots, m-1$. Then, the set of weight vectors, $\Lambda = \{\lambda^{[\mathbf{h}]} = \frac{\mathbf{h}}{H} | \sum_{i=0}^{m-1} h_i = H\}$, can be

called a uniform spread of weight vectors where **h** is referred to as the m-dimensional index of weight vector $\lambda^{[\mathbf{h}]} \in \Lambda$.

In the overlapped partitioning island model based on weight vectors, the entire population for a bi-objective optimization problem (BOP) is first evenly split into several non-overlapping partitions, each of which evolves separately on an individual processor in a cluster. Let N denote the size of the entire population P and q be the number of participating processors. Then, the rth partition $P^{(r)}$, $r \in [0..q-1]$, has size $N^{(r)} = N/q + 1$ if $r < (N \ mod \ q)$, and otherwise $N^{(r)} = N/q$, where mod indicates the modulo operation.

Besides, a certain number of overlapping individuals from the neighbor partitions are also appended to each end of a partition in the overlapped partitioning island model so that the entire operation of updating neighbor individuals in the MOEA/D can be completed independently within the respective processor. A partition $P^{(r)}$ and its corresponding overlapping portions constitute a separate subpopulation or island $P^{\{r\}}$, which is in charge of the evolution of individuals only within its partition $P^{(r)}$. Since the uniformly distributed weight vectors in the bi-objective space obey a one-dimensional linear ordering, it is relatively easy to determine the overlapping portions of any subpopulation for BOPs. Figure 1 presents the overlapped partitioning island model for BOPs for the simple case where f_0 and f_1 are two objectives to be minimized, the population size $N = 10$, the number of divisions for each dimension $H = N - 1 = 9$, the number of participating processors $q = 5$ and the size of the overlapping portion $k = 1$. By using overlapped partitioning, the sizes of the resulting partitions are $N^{(r)} = 2$, $r = 0, 1, 2, 3, 4$, and the resulting subpopulations are $P^{\{r\}} = \{P_i | i \in [0..9] \cap [2r-1..2r+2]\}$, where P_i represents the ith individual in charge of optimizing the ith subproblem with weight vector $\lambda^{[(H-i,i)]} = (\frac{H-i}{H}, \frac{i}{H})$. As a result, every partition is responsible for converging towards a segment of the Pareto front (PF). It is worth noting that the first partition $P^{(0)}$ has k overlapping individuals appended to its rightmost side only, the last partition $P^{(q-1)}$ has k overlapping individuals appended to its leftmost side only and any other partition $P^{(r)}, r \in [1..q-2]$, has $2 \times k$ overlapping individuals, respectively, appended to its leftmost and rightmost sides as shown in the example in Fig. 1.

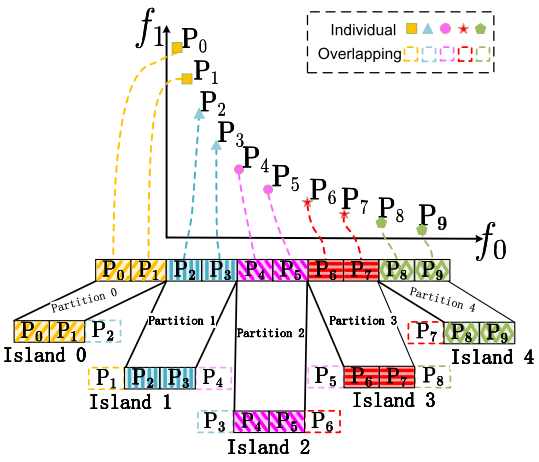

Fig. 1 Overlapped partitioning island model

2.2 Elitist policy and linear topology for migration

The overlapping individuals in a subpopulation may be better than the corresponding individuals in the neighbor subpopulations in the overlapped partitioning island model. Thus, subpopulations on separate processors must cooperate to speed up convergence using an elitist migration policy. The elitist policy helps share the results of evolution between neighbor subpopulations and accelerates convergence by regularly exchanging the updated overlapping individuals at each side of each subpopulation, referred to as elitist individuals. Owing to the one-dimensional linear ordering between individuals with assigned weight vectors for BOPs, elitist individuals migrate according to a linear topology where the neighbors of a subpopulation include only its preceding and succeeding subpopulations. Specifically, elitist individuals at both sides of any subpopulation $P^{\{r\}}$, $r \in [0..q-1]$, emigrate to its preceding neighbor subpopulation $P^{\{r-1\}}$, $r \in [1..q-1]$, and its succeeding one $P^{\{r+1\}}$, $r \in [0..q-2]$, as shown by the linear topology in Fig. 2. Every elitist individual immigrating to subpopulation $P^{\{r\}}$, $r \in [0..q-1]$, is used to update its neighbor individuals in the local subpopulation $P^{\{r\}}$ in the same way as the update operator for neighbors in the MOEA/D.

2.3 Procedure for the opMOEA/D

The procedure for the opMOEA/D using the island model with the elitist policy and linear topology discussed above is presented in Algorithm 1. The key idea behind the opMOEA/D is that it distributes evolution of the population in the MOEA/D among q concurrent processes running on q separate processors in lines 1–22 of the opMOEA/D according to the overlapped partitioning island model. A sub-

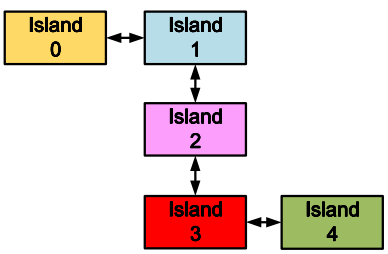

Fig. 2 Linear topology for migration

Algorithm 1 The main procedure of opMOEA/D

```
1: for each r ∈ [0..q − 1] in parallel do                ▷ q processors
2:     N^{r} ← initializeSubpopSize(N, q, r)
3:     Λ^{r} ← initializeSubpopWeight(r)
4:     P^{r}, P^{(r)} ← initializeSubpop(Λ^{r})
5:     B ← initializeSubpopNeighborhood(Λ^{r})
6:     A^{r} ← initializeNeighborIslands(r)
7:     z ← initializeInternalUtopianPoint(P^{r})
8:     z ← synchronizeUtopianPoint(z)
9:     gen ← 0
10:    while gen≤MaxGen do                    ▷ evolutionary loop
11:        for each p ∈ P^{(r)} do
12:            parents ← selection(P^{r}, p)
13:            child ← crossover(parents)
14:            child ←mutation(child)
15:            evaluateFitness(child)
16:            internalUtopianPointUpdate(child, z)
17:            internalSolutionsUpdate(child, p, P^{r})
18:        end for
19:        elitistMigration(P^{r}, P^{(r)}, A^{r})
20:    end while
21:    P ←synchronizePop(P^{(r)})
22: end for
```

population is first initialized in lines 2–7 in each process. The function in line 2 generates the size $N^{\{r\}}$ of the subpopulation $P^{\{r\}}$ in process r, while that in line 3 assigns a weight vector to each individual within $P^{\{r\}}$. In line 6, the subpopulation $P^{\{r\}}$ maintains a list $A^{\{r\}}$ of all its neighbor subpopulations according to the linear topology. Before evolution, root process 0 gathers the external utopian points from all the other processes to update its internal utopian point and then broadcasts this back in line 8. In the evolutionary loop in lines 10–20, the major difference from the sequential MOEA/D is that the separate processes of the opMOEA/D must cooperate through the elitist migration policy in line 19, the pseudo-code for which is given in Algorithm 2. After evolution, root process 0 gathers the q partitions from all processes to compose the final population P in line 21.

In Algorithm 2, the elitist migration policy in lines 2–10 is responsible for emigrating each elitist individual in a subpopulation to its neighbor subpopulations according to the linear topology every ten generations since the migration traffic of elitists is fairly heavy. Besides, process r receives the elitist individuals from each neighbor subpopulation in $A^{\{r\}}$ and updates all their neighbor individuals in the local subpopulation $P^{\{r\}}$ in line 9.

3 Partially evolved parallelization of the MOEA/D for multi-objective optimization

Although the task to determine and maintain overlapping individuals for a subpopulation in the overlapped partitioning MOEA/D is linear and easy in the bi-objective space,

Algorithm 2 Elitist migration policy

```
1: procedure elitistMigration(P^{r}, P^{(r)}, A^{r})
2:     if gen%10 == 0 then                ▷ elitist migration
3:         for each p ∈ P^{r} \ P^{(r)} do
4:             if p.isIndivUpdated then
5:                 sendElitistIndiv(p, A^{r})
6:                 p.isIndivUpdated ← false
7:             end if
8:         end for
9:         externalSolutionsUpdate(P^{r}, A^{r})
10:    end if
11: end procedure
```

it generally becomes nonlinear and very difficult in three or higher dimensional objective spaces. As a result, the opMOEA/D is only suitable for bi-objective optimization. Besides, the topology for migration in the opMOEA/D is fixed and not flexible enough because an elitist individual with weight vector $\lambda^{[\mathbf{h}]}$ in a subpopulation can only emigrate to a few fixed neighbor subpopulations that also contain the individual with $\lambda^{[\mathbf{h}]}$. Thus, another universal partially evolved MOEA/D, referred to as the peMOEA/D, is proposed to extend the opMOEA/D from the bi-objective space to three or higher dimensional objective spaces by introducing a partially evolved island model and a nearest-neighbor partitioning approach as well as a hybrid migration policy and an adaptive neighbor-based topology.

3.1 Partially evolved island model and nearest-neighbor partitioning approach

The critical idea underlying the partially evolved island model is that it keeps a subpopulation equal in size to the entire population, but only evolves a partial (equal to a partition size) subpopulation, rather than the entire subpopulation, on each separate processor in a cluster. The other individuals in the subpopulation are used to complete the operation of updating neighbor individuals independently on the respective processor. As a result, this model has the primary advantage of avoiding the complicated task of maintaining overlapping individuals in a subpopulation in three or higher dimensional objective spaces. The size $N^{(r)}$ of partition $P^{(r)}$ in processor r is determined by a similar rule in the overlapped partitioning island model, i.e., $N^{(r)} = N/q + 1$ if $t < (N \bmod q)$, and otherwise, $N^{(r)} = N/q$. The partitions can easily be obtained by splitting the individuals linearly in the bi-objective space. Figure 3 illustrates the partially evolved island model over five processors in the bi-objective space where the population size $N = 10$, the number of divisions for each dimension $H = N - 1 = 9$, the number of participating processors $q = 5$, the sizes of the resulting subpopulations $N^{\{r\}} = 10$, $r = 0, 1, 2, 3, 4$, the sizes of the resulting partitions $N^{(r)} = 2$ and the resulting partitions are $P^{(r)} = \{P_i \mid i \in [2r..2r + 1]\}$.

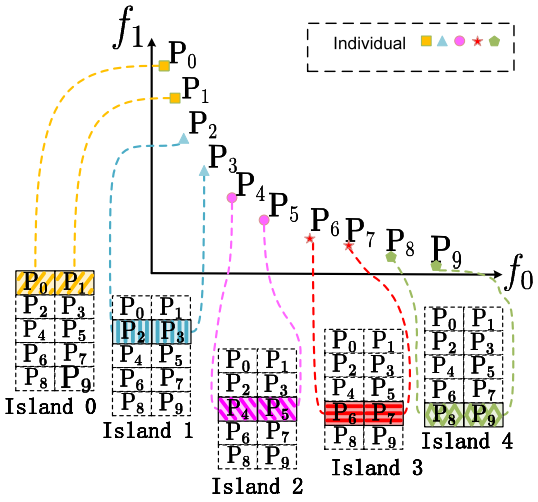

Fig. 3 Partially evolved island model

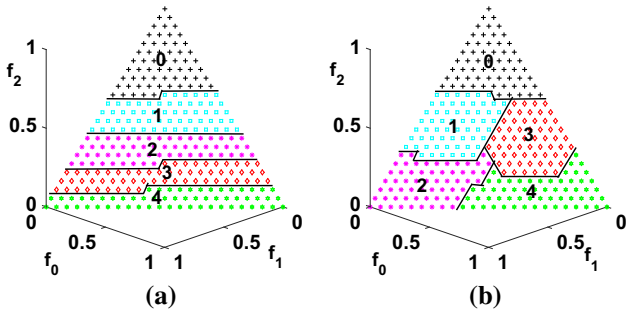

Fig. 4 Partitions generated, respectively, by **a** the linear partitioning approach and **b** the nearest-neighbor partitioning approach in the three-dimensional objective space

However, the weight vectors generally distribute nonlinearly in three or higher dimensional objective spaces. In these cases, a simple linear partitioning approach forcibly arranges the weight vectors in lexicographical order and then splits them linearly. Figure 4a shows the partitions generated by linear partitioning of the weight vectors (or the corresponding weighted individuals) in the three-dimensional objective space with the number of partitions $q = 5$. Unfortunately, linear partitioning often generates many very "flat" partitions where the maximal differences in some dimensions of the weight vectors are very small while those in other dimensions are extremely large. A flat partition usually leads to a high probability of the neighbors of its individuals going beyond it and heavy migration traffic of elitist individuals.

Definition 2 (*Nearest-neighbor vectors*) Let $\lambda^{[\mathbf{h}]} \in \Lambda$ be the weight vector for one subproblem and $\mathbf{h}^{(a,b)} = (h_0^{(a,b)}, \ldots, h_{m-1}^{(a,b)})$, $a \in [0..m-1]$, $b \in [0..m-1]$, $a \neq b$, where $h_i^{(a,b)} = h_i$ except that $h_a^{(a,b)} = h_a - 1$ and $h_b^{(a,b)} = h_b + 1$. Then, the weight vectors $\lambda^{[\mathbf{h}^{(a,b)}]} \in \Lambda$ are the nearest-neighbor to weight vector $\lambda^{[\mathbf{h}]}$.

Algorithm 3 Nearest-neighbor partitioning

```
1:  function nearestNeighborPartitioning(q, Λ)
2:      Λ†←∅; P♯←∅; l ← 0;
3:      for each r ∈ [0..q − 1] do          ▷ q partitions
4:          if l == 0 then
5:              root←lexicoMin(Λ − Λ†); l←λ_{m−1}
6:          else
7:              root ← lexicoMin(Q)
8:          end if
9:          Q←∅; append root into the tail of
    queue Q
10:         P^(r)←∅; n ← 0                  ▷ begin partitioning
11:         while n < N^(r) do
12:             λ ← Q.head(); Q.removeHead()
13:             if λ ∈ Λ then
14:                 add λ to P^(r) and Λ†
15:                 l← Min{l, λ_{m−1}}; n++
16:                 for each λ′ ∈Nearest(λ) do
17:                     if λ′∉Λ†∧λ′∉Q then
18:                         append λ′ into the tail
    of Q
19:                     end if
20:                 end for
21:             end if
22:         end while
23:         P♯←P♯ ∪ {P^(r)}
24:     end for
25:     return P♯
26: end function
```

3.2 Hybrid policy and adaptive neighbor-based topology for migration

Another nearest-neighbor partitioning approach is also introduced based on a breadth-first search of the graph of the nearest-neighbor relationship of weight vectors in Definition 2 in this paper. Definition 2 indicates that the m-dimensional indexes of two nearest-neighbor weight vectors have only two different components, and weight vector $\lambda^{[\mathbf{h}]}$ has at most $2C_m^2$ nearest-neighbor weight vectors. Algorithm 3 gives the pseudo-code for the nearest-neighbor partitioning. A root vector for the next breadth-first search is chosen in lines 4–9. The control variable l in line 15 is used to choose an appropriate root vector by recording the minimum in the last dimension of the searched weight vectors. This will be reset when it reaches zero in line 7. Function Nearest(λ) in line 16 returns all nearest-neighbor vectors of λ, while function lexicoMin(Q) in line 7 returns the minimal weight vector in lexicographical order from a given set Q. The breadth-first search from a root vector is performed once every loop iteration in lines 10–22 to yield a "fatter" partition $P^{(r)}$ than that obtained by linear partitioning. Figure 4b presents the

partitions generated by nearest-neighbor partitioning of the weight vectors in the three-dimensional objective space. A fatter partition usually means a lower probability of the neighbors of its individuals going beyond it and lighter migration traffic.

Owing to the distributed memory in message-passing clusters, each processor must maintain its own local utopian point, which in fact greatly influences the search for the front segment in the charge of its own subpopulation. Therefore, a hybrid policy including both elitist and utopian migration is adopted in the peMOEA/D to help subpopulations cooperate effectively to improve their local utopian points and individuals across separate processors. Utopian migration can propagate better utopian points throughout subpopulations and widen the search scope for the front segment in the charge of each subpopulation. A utopian point, often called an ideal point, is composed of the best attainable value out of the given solution set for each objective. A better utopian point can assist the population of the MOEA/D to approximate a wider PF. Since a separate utopian point must be maintained for any subpopulation in the peMOEA/D, the updated utopian point for any subpopulation $P^{\{r\}}$ emigrates to its neighbor subpopulations. A utopian point immigrating to any subpopulation $P^{\{r\}}$ helps improve the local utopian point for $P^{\{r\}}$ according to the update operator of the utopian point in the MOEA/D. The hybrid migration policy includes the advantages of both pure policies. Migration of elitist individuals is performed in the peMOEA/D in a similar manner to that in the opMOEA/D.

Definition 3 (*Neighbor partitions*) Let $P^{(i)}$ and $P^{(j)}, i \neq j$, be the partitions in two subpopulations $P^{\{i\}}$ and $P^{\{j\}}$ in the partially evolved island model, respectively. Then, $P^{(i)}$ and $P^{(j)}$ are neighbor to each other if and only if there exist $p \in P^{(i)}$ and $p' \in P^{(j)}$ and p is the nearest-neighbor to p'.

Benefiting from the partially evolved island model, every weight vector (or its corresponding weighted individual) exists in a separate subpopulation, which means that elitist individuals can be migrated to any subpopulation. Thus, several flexible topologies, such as the linear topology and the complete graph topology shown in Figs. 2 and 5a, respectively, are available in the peMOEA/D for hybrid migration. Nevertheless, these often lead to either too light or too heavy migration traffic. This paper presents an adaptive neighborhood-based topology for migration based on the neighbor relationship between partitions given in Definition 3. A pair of subpopulations is connected in this topology if and only if their corresponding partitions are neighbors. Figure 5b illustrates the adaptive neighborhood-based topology for the partitions shown in Fig. 4b. Thus, evolutionary information can be directly transferred to the neighbor partitions with fewer transitive processors in this topology than in the linear one, while it exhibits lower migration traffic and better scalability than the complete graph topology.

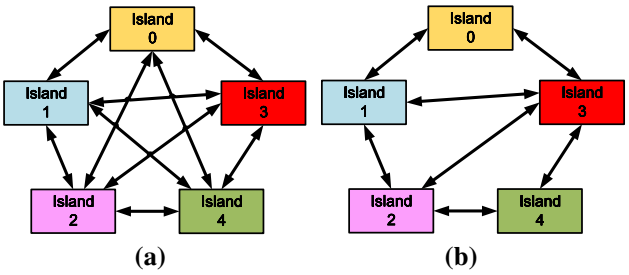

Fig. 5 Illustration of **a** the complete graph topology and **b** the adaptive neighbor-based topology for migration

3.3 Procedure for the peMOEA/D

The procedure for the peMOEA/D is presented in Algorithm 4. An entire population is initialized on each processor in line 4. The function in line 2 generates the size $N^{(r)}$ of partition $P^{(r)}$ on processor r. Then, the set $P^{\sharp}$ of resulting partitions $P^{(r)}$ is generated in line 6. The function in line 7 returns the list $A^{\{r\}}$ of neighbor islands of $P^{\{r\}}$ in the adaptive neighbor-based migration topology. According to the partially evolved island model, only individuals in partition $P^{(r)}$ participate in the evolutionary loop on processor r in lines 12–19. The procedure in line 20 exchanges evolutionary information including elitist individuals and utopian points across separate processors using the hybrid migration policy, the pseudo-code for which is given in Algorithm 5. The other steps in this procedure are similar to those for the opMOEA/D in Algorithm 1 except that the size of the subpopulation is fixed at $N^{\{r\}} = N$ on all processors in the peMOEA/D.

In Algorithm 5, hybrid migration includes both elitist migration and utopian migration, both of which obey the adaptive neighbor-based topology in the peMOEA/D. The procedure for elitist migration in the peMOEA/D is similar to that in Algorithm 2. The utopian point migration policy is responsible for emigrating the updated utopian point to all neighbor islands $A^{\{r\}}$ in lines 6–10 and updating the local utopian point by receiving the utopian points from all neighbor islands $A^{\{r\}}$ according to the adaptive neighbor-based topology.

4 Experimental results and analysis

In this section, we discuss several experiments on some widely used benchmark problems with two, three and five objectives conducted to validate the effectiveness of our proposed parallel algorithm, peMOEA/D. All the simulations were carried out on a message-passing cluster composed of

Algorithm 4 The main procedure of peMOEA/D

```
1: for each r ∈ [0..q − 1] in parallel do                ▷ q processors
2:     N^(r) ← initializePartiallyEvolvedSize(N, q, r)
3:     Λ ← initializePopulationWeight( )
4:     P^{r} ← initializePopulation( )
5:     B ← initializeWeightNeighborhood(Λ)
6:     P^♯ ← nearestNeighborPartitioning(q, Λ)
7:     A^{r} ← initializeNeighborIslands(P^♯, Λ)
8:     z ← initializeLocalUtopianPoint(P)
9:     z ← synchronizeUtopianPoint(z)
10:    gen ← 0
11:    while gen≤MaxGen do ▷ evolutionary loop
12:        for each p ∈ P^(r) do
13:            parents ← selection(P^{r}, p)
14:            child ← crossover(parents)
15:            child ←mutation(child)
16:            evaluateFitness(child)
17:            internalUtopianPointUpdate(child, z)
18:            internalSolutionsUpdate(child, p, P^{r})
19:        end for
20:        hybridMigration(P^{r}, P^(r), A^{r}, z)
21:    end while
22:    P ←synchronizePop(P^(r))
23: end for
```

Algorithm 5 Hybrid migration policy

```
1: procedure hybridMigration(P^{r}, P^(r), A^{r}, z)
2:     elitistMigration(P^{r}, P^(r), A^{r})
3:     utopianMigration(z, A^{r})
4: end procedure
5: procedure utopianMigration(z, A^{r})
6:     if z.isUtopianUpdated then                ▷ utopian migration
7:         sendUtopianPoint(z, A^{r})
8:         z.isUtopianUpdated ← false
9:     end if
10:    z ←externalUtopianPointUpdate(z, A^{r})
11: end procedure
```

20 computing nodes equipped with Intel Core I5 3.20 GHz CPU and 4 GB RAM, which are fully interconnected by a high-speed Gigabit Ethernet switch. All the sequential and parallel MOEA/Ds were implemented in C++ using MPI. An aggregation function of penalty-based boundary intersection (Zhang and Li 2007; Li and Zhang 2009) with reliant parameter $\theta = 5$ was adopted for each sequential and parallel MOEA/D. The crossover and mutation operators used in each sequential and parallel MOEA/D in this paper are the same as those in (Ying et al. 2012; Ying and Xu 2012). The neighborhood size of subproblems was set to $T = 20$ in our experiments. Besides, the inverted generational distance (IGD) metric used in (Zhang and Li 2007; Ying et al. 2012) was also adopted to assess the quality of solutions discovered by each algorithm for each benchmark problem in our experiments. The IGD metric measures how far the elements in the true Pareto front are away from a set of discovered solutions in the objective space. Therefore, a lower value for IGD metric usually denotes better quality of the set of discovered solutions. For each problem instance, 30 statistically independent runs of each algorithm were conducted in our experiments.

4.1 Experiments on BOPs

Since the opMOEA/D is only suitable for bi-objective optimization, the peMOEA/D was first tested and compared with the sequential MOEA/D and opMOEA/D, peMOEA/D$_e$ and peMOEA/D$_u$ on five widely used benchmark BOPs (Zitzler et al. 2000), namely ZDT$_{1-3}$ and ZDT$_{4,6}$, where peMOEA/D$_e$ and peMOEA/D$_u$ denote two variants of the peMOEA/D using the pure elitist migration policy and the pure utopian one, respectively. It should be noted that only the nearest-neighbor partitioning approach and the adaptive neighbor-based topology were used in all variants of the peMOEA/D because these are, respectively, equivalent to the linear partitioning approach and linear topology in the bi-objective space discussed in this subsection. Thirty decision variables were used for ZDT$_{1-3}$, while ZDT$_{4,6}$ were tested with ten decision variables in our experiments. For the sake of fairness, each sequential and parallel MOEA/D used a population size $N = 200$ and terminated when the number of evolution generations reached 600 for each benchmark BOP. The size of the overlapping portion at each side of a subpopulation was set to $k = \frac{1}{2}T$ in the opMOEA/D.

The number of participating processors in the cluster was first set to $q = 8$. Figure 6 shows the convergence curves of the average IGD scores (on a logarithmic scale) over 30 runs for each sequential and parallel MOEA/D for each benchmark problem on the cluster. Figure 7 plots the front segment corresponding to each partition discovered, respectively, by the peMOEA/D$_e$, peMOEA/D$_u$ and peMOEA/D in the run with the lowest IGD value for ZDT$_2$. Figure 6 indicates that the peMOEA/D$_e$ and opMOEA/D, both of which use only the pure elitist migration policy, achieve nearly the same performance for these BOPs. It can be seen from Fig. 6 that the peMOEA/D$_e$ usually converges faster in the early and middle stages of evolution than the peMOEA/D$_u$ for these BOPs, which suggests that the migration of elitist individuals can help accelerate evolutionary convergence to some extent. However, it is also evident from Fig. 6 that the peMOEA/D$_u$ often achieves better fronts in the terminal stage of evolution than the peMOEA/D$_e$ does for five BOPs. This is attributed to the fact that the migration of utopian points helps guide the evolutionary search towards a wider front by propagating the better utopian points throughout separate subpopulations.

Furthermore, as demonstrated for ZDT$_2$ in Fig. 7, the peMOEA/D$_e$ without migration of utopian points often converges towards several narrow front segments and obtains an incomplete front that usually breaks between neighbor partitions owing to worse utopian points on some individ-

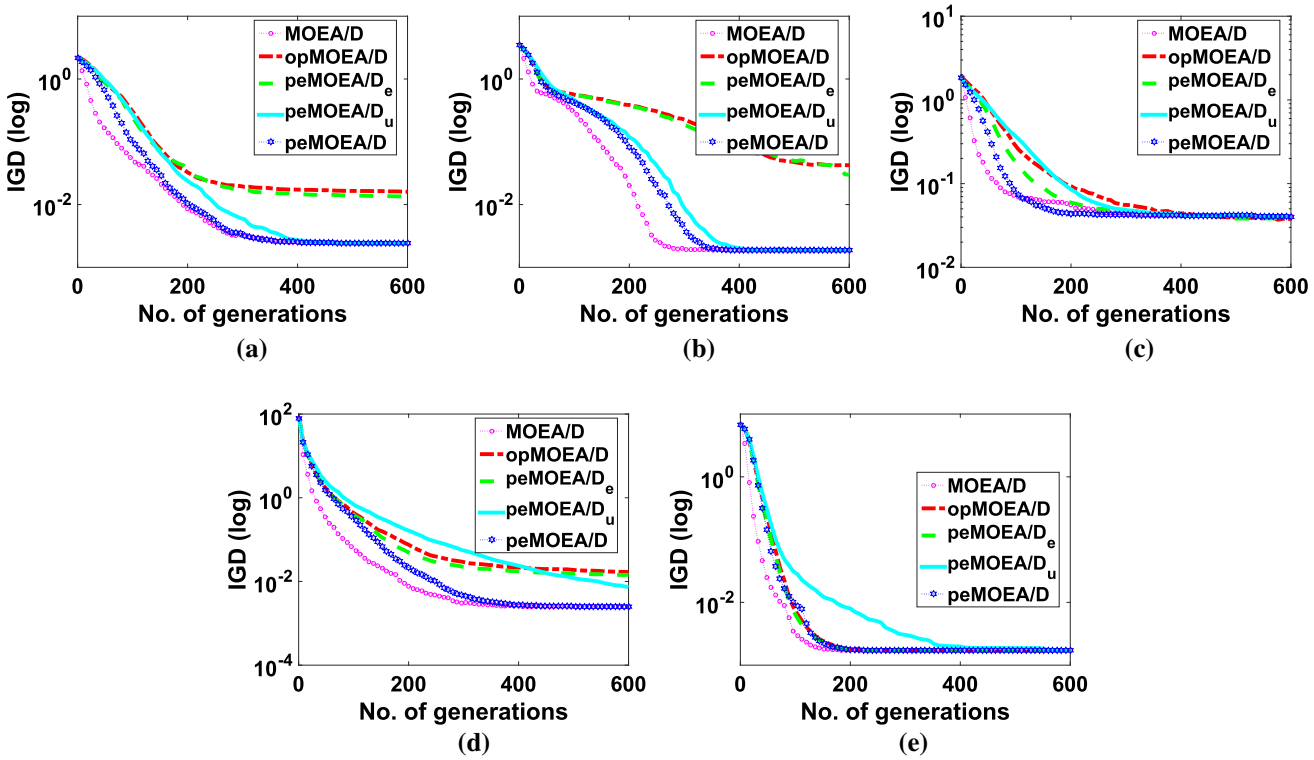

Fig. 6 Convergence curves of average IGD scores over 30 runs for each sequential and parallel MOEA/D for **a** ZDT_1, **b** ZDT_2, **c** ZDT_3, **d** ZDT_4 and **e** ZDT_6

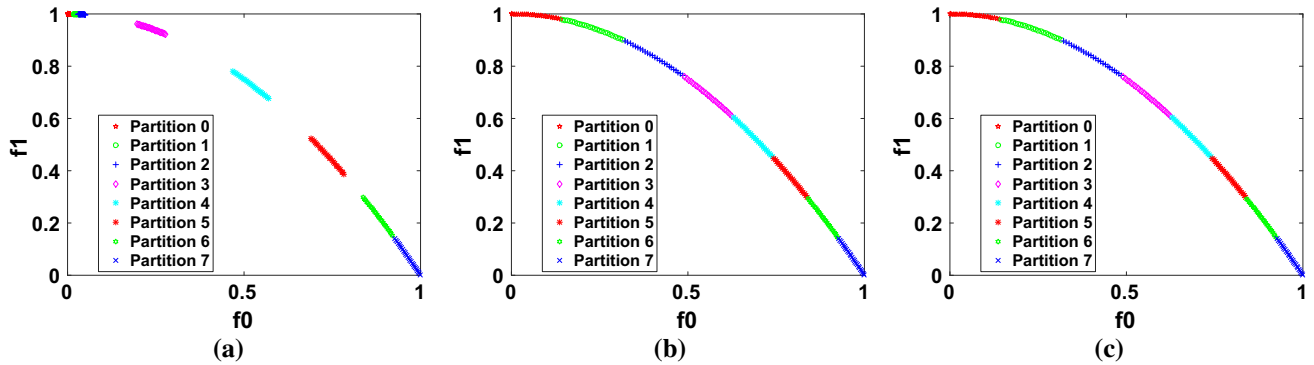

Fig. 7 Front segment corresponding to each partition discovered, respectively, by **a** peMOEA/D_e, **b** peMOEA/D_u and **c** peMOEA/D for ZDT_2

ual processors. Meanwhile, it can be inferred from Figs. 6 and 7 that the peMOEA/D and peMOEA/D_u converge to nearly the same complete front as the sequential MOEA/D by propagating utopian points. Figure 6 also indicates that the peMOEA/D not only converges faster than the peMOEA/D_u does in the early stage, but also achieves more complete fronts than the peMOEA/D_e does in the terminal stage. Consequently, it can be concluded that the hybrid migration policy helps subpopulations on separate processors to cooperate more effectively than either of the pure migration policies does.

Figure 8 shows the speedup curves achieved by each parallel MOEA/D with an increase in participating processors from one to eight for the benchmark BOPs on the cluster. It indicates that all parallel MOEA/Ds can clearly reduce computation time for all five BOPs with the peMOEA/D achieving very slightly lower speedups than the other three parallel MOEA/Ds despite using the hybrid policy to migrate both elitist individuals and utopian points. In conclusion, the peMOEA/D clearly performs better in terms of quality of solutions than the opMOEA/D and peMOEA/D_e, and converges faster than peMOEA/D_u, although these algorithms achieve similar speedups for the benchmark BOPs. The experimental results on BOPs show that the first primary advantage of the peMOEA/D over the opMOEA/D is that the hybrid migration policy clearly yields better overall

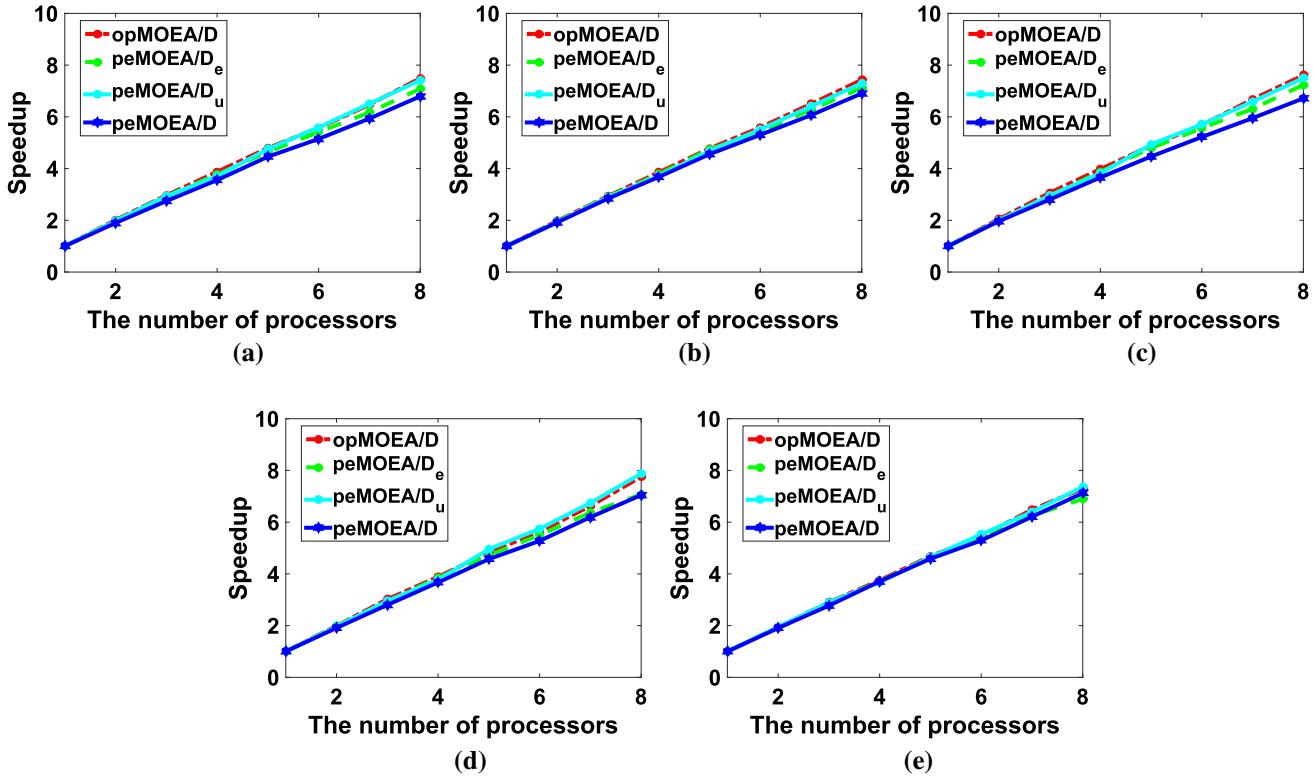

Fig. 8 Speedup curves achieved by each parallel MOEA/D with an increase in processors for **a** ZDT_1, **b** ZDT_2, **c** ZDT_3, **d** ZDT_4 and **e** ZDT_6

performances at the cost of very slight speedup degradation. Therefore, the default hybrid migration policy was adopted for the peMOEA/D and its variants in the subsequent experiments on MOPs unless specified otherwise.

4.2 Experiments on MOPs with three or more objectives

Another primary advantage of the peMOEA/D over the opMOEA/D is the universality due to the partially evolved island model, which means that the peMOEA/D is suitable not only for BOPs but also for MOPs with three or more objectives. To assess the overall performance of the peMOEA/D using the nearest-neighbor partitioning approach and adaptive neighbor-based topology in higher dimensional objective spaces, it was further compared with the sequential MOEA/D and four variants of the peMOEA/D, denoted as peMOEA/D^{ll}, peMOEA/D^{nl}, peMOEA/D^{lc}, and peMOEA/D^{nc}, for three- and five-objective $DTLZ_{1-4}$ (Deb et al. 2002b; Deb and Jain 2014), where l and n as the leftmost superscript indicate, respectively, that the linear partitioning approach and nearest-neighbor partitioning approach were adopted, and l and c as the rightmost superscript denote, respectively, that the linear migration topology and complete graph migration topology were used in the variants. In our experiments, we set the numbers of decision variables to $n = m + k - 1$ with $k = 5$ for $DTLZ_1$ and $k = 10$ for $DTLZ_{2-4}$ and m denoting the number of objectives. The population sizes for the three- and five-objective DTLZ problems were set to $N = C_{H+m-1}^{m-1} = C_{23+3-1}^{3-1} = 300$ and $N = C_{H+m-1}^{m-1} = C_{8+5-1}^{5-1} = 495$, respectively. Each sequential and parallel algorithm terminated after the evolution of 400 generations for each three- and five-objective DTLZ problem.

The number of participating processors was first set to $q = 8$. Figures 9 and 10 show the convergence curves of the average IGD scores (on a logarithmic scale) over 30 runs for each sequential and parallel MOEA/D for three- and five-objective $DTLZ_{1-4}$, respectively. Figure 11 shows the box-plots of the IGD scores obtained by each sequential and parallel MOEA/D over 30 runs for three-objective $DTLZ_{3,4}$. Figure 12 presents the front segment corresponding to each partition discovered by each sequential and parallel MOEA/D in the run with the worst IGD value for three-objective $DTLZ_4$.

It is evident from Figs. 9 and 10 that all variants of the peMOEA/D clearly achieve better quality solutions in terms of IGD scores than the sequential MOEA/D does for $DTLZ_{3,4}$ with three or five objectives while achieving nearly the same performance as the sequential MOEA/D for relatively simple $DTLZ_{1,2}$ with three or five objectives. It should be noted that there are many local Pareto-optimal fronts parallel to the true PF for $DTLZ_3$. As indicated in Fig. 11a,

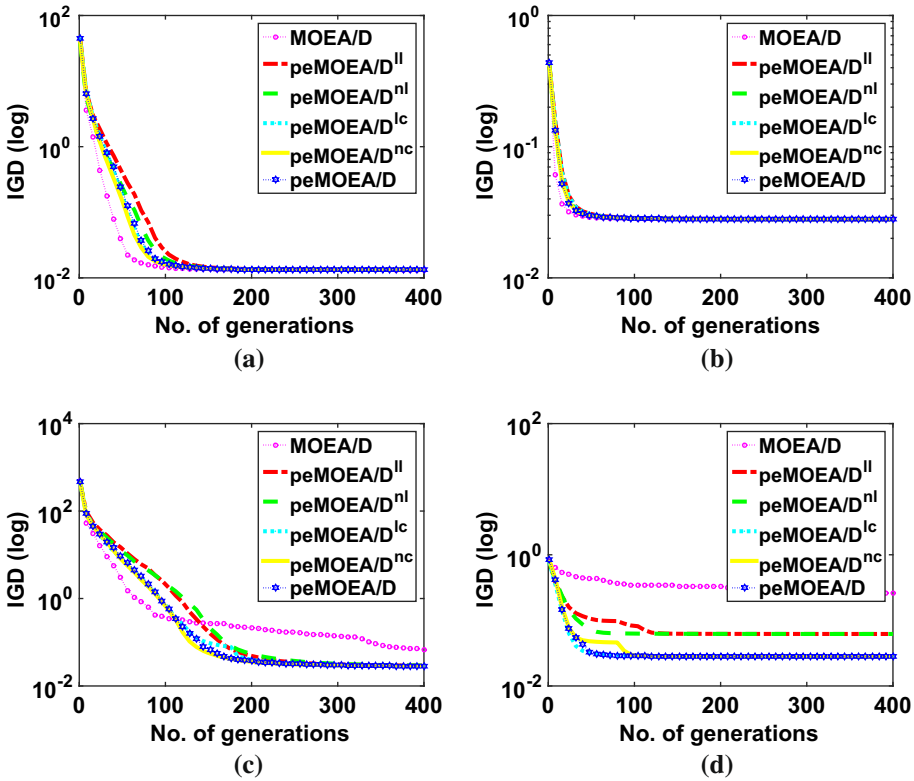

Fig. 9 Convergence curves of average IGD scores over 30 runs for each sequential and parallel MOEA/D for three-objective **a** $DTLZ_1$, **b** $DTLZ_2$, **c** $DTLZ_3$ and **d** $DTLZ_4$

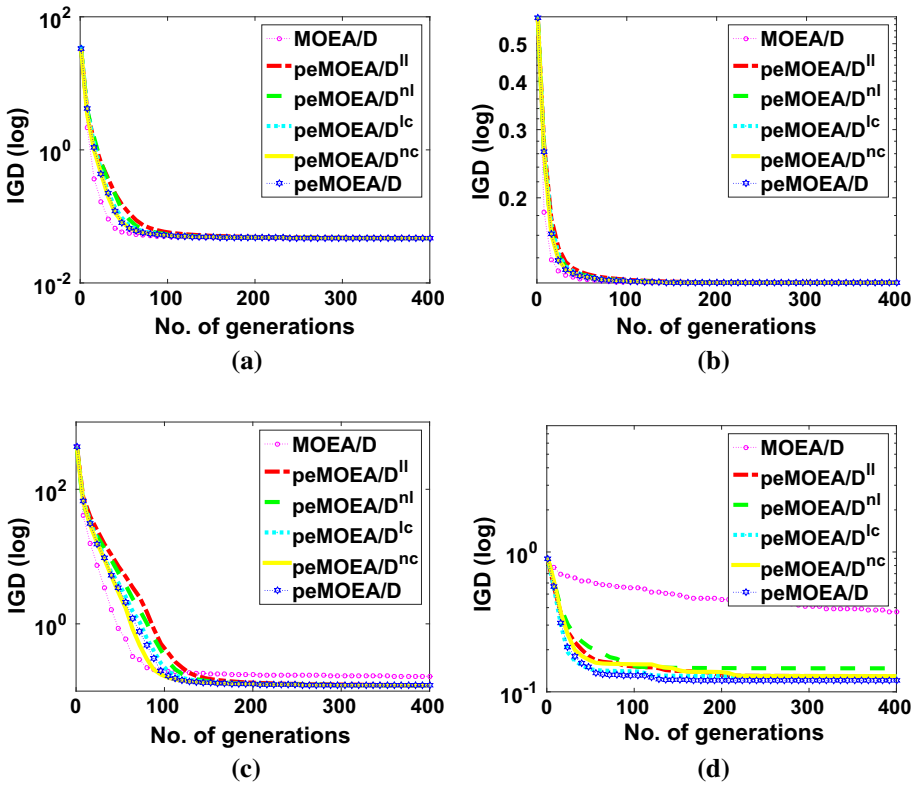

Fig. 10 Convergence curves of average IGD scores over 30 runs for each sequential and parallel MOEA/D for five-objective **a** $DTLZ_1$, **b** $DTLZ_2$, **c** $DTLZ_3$ and **d** $DTLZ_4$

whereas the sequential MOEA/D gets stuck at local fronts in three out of the 30 runs for three-objective $DTLZ_3$, each variant of the peMOEA/D successfully escapes these in all 30 runs in our experiments owing to the increased diversity of the subpopulations with a large size N in the partially evolved island model.

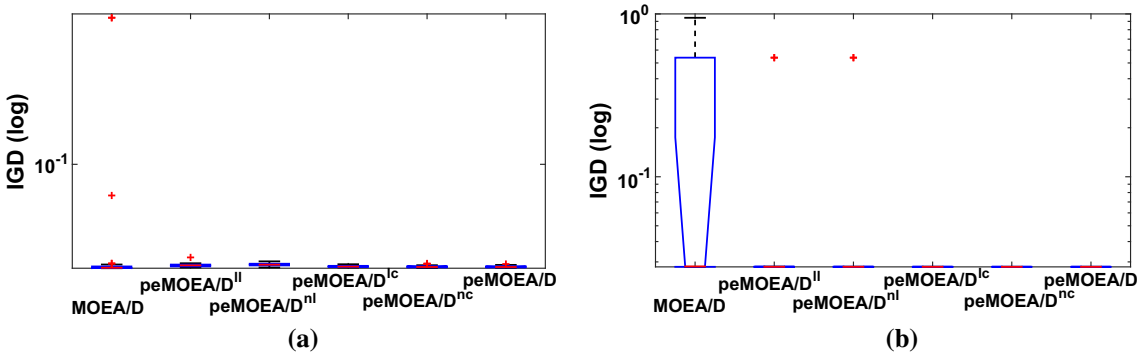

Fig. 11 Box-plots of IGD scores obtained by each sequential and parallel MOEA/D over 30 runs for three-objective **a** $DTLZ_3$ and **b** $DTLZ_4$

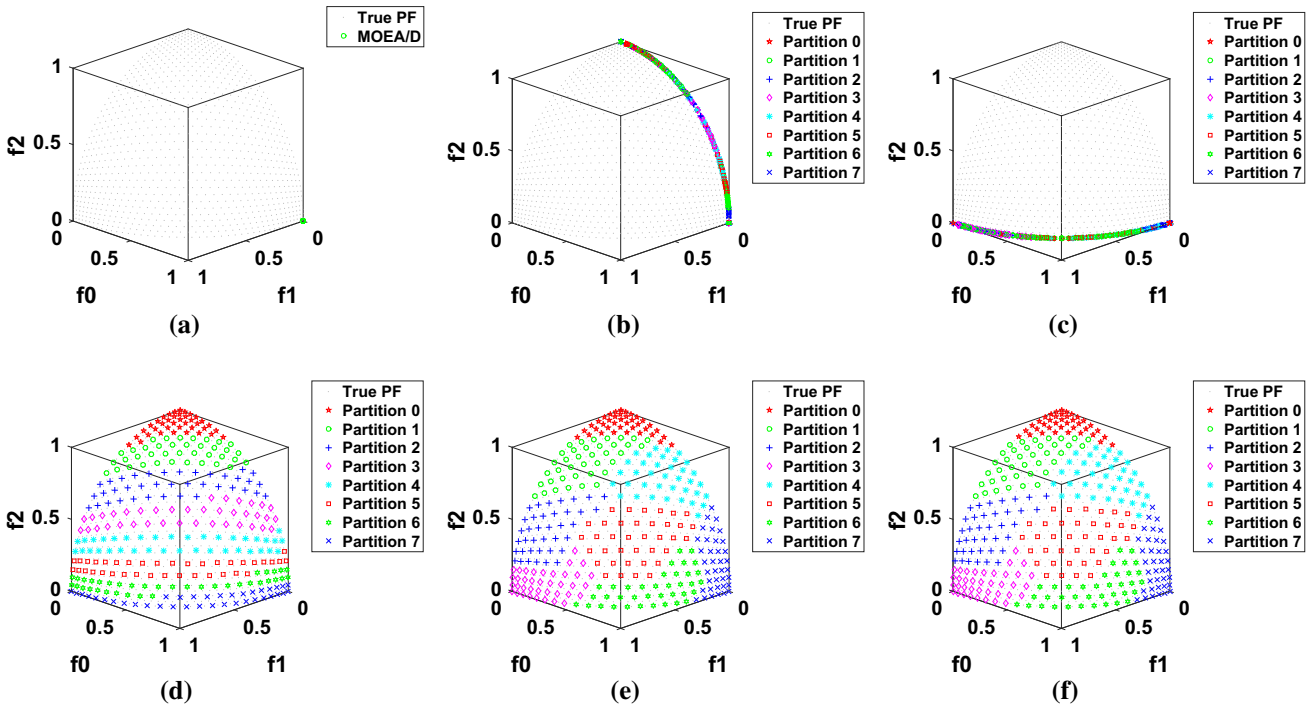

Fig. 12 Front segment corresponding to each partition discovered, respectively, by **a** MOEA/D, **b** peMOEA/D^{ll}, **c** peMOEA/D^{nl}, **d** peMOEA/D^{lc}, **e** peMOEA/D^{nc} and **f** peMOEA/D for three-objective $DTLZ_4$

Furthermore, it can also be seen from Figs. 9 and 10 that the peMOEA/D^{ll} has nearly the same convergence performance in terms of IGD scores as the peMOEA/D^{nl} for three- and five-objective $DTLZ_{1-4}$ while the peMOEA/D^{lc} has nearly the same as the peMOEA/D^{nc}. At the same time, the peMOEA/D as well as the peMOEA/D^{lc} and peMOEA/D^{nc} converges faster than the peMOEA/D^{ll} and peMOEA/D^{nl}. Additionally, it can be inferred from Fig. 11b that the peMOEA/D, peMOEA/D^{lc} and peMOEA/D^{nc} have no difficulty in obtaining a complete front for three-objective $DTLZ_4$ despite the extremely variable densities of solutions in the objective space. However, the sequential MOEA/D converges only to the anchor point with the highest density in 12 out of 30 runs in our experiments, as intuitively demonstrated in Fig. 12. Meanwhile, the peMOEA/D^{ll} and peMOEA/D^{nl} converges only to the boundary curve with a very high density along the true PF in three out of 30 runs and two out of 30 runs, respectively, for three-objective $DTLZ_4$. This implies that the migration topology is critical for the convergence performance of the peMOEA/D and both the adaptive neighbor-based topology in the peMOEA/D and the complete graph migration topology in the peMOEA/D^{lc} and peMOEA/D^{nc} have higher degrees of connectivity and a better ability to propagate elitist individuals and utopian points across separate subpopulations than the linear topology in the peMOEA/D^{ll} and peMOEA/D^{nl}.

Figures 13 and 14 show the average speedup curves achieved by each variant of peMOEA/D with an increase

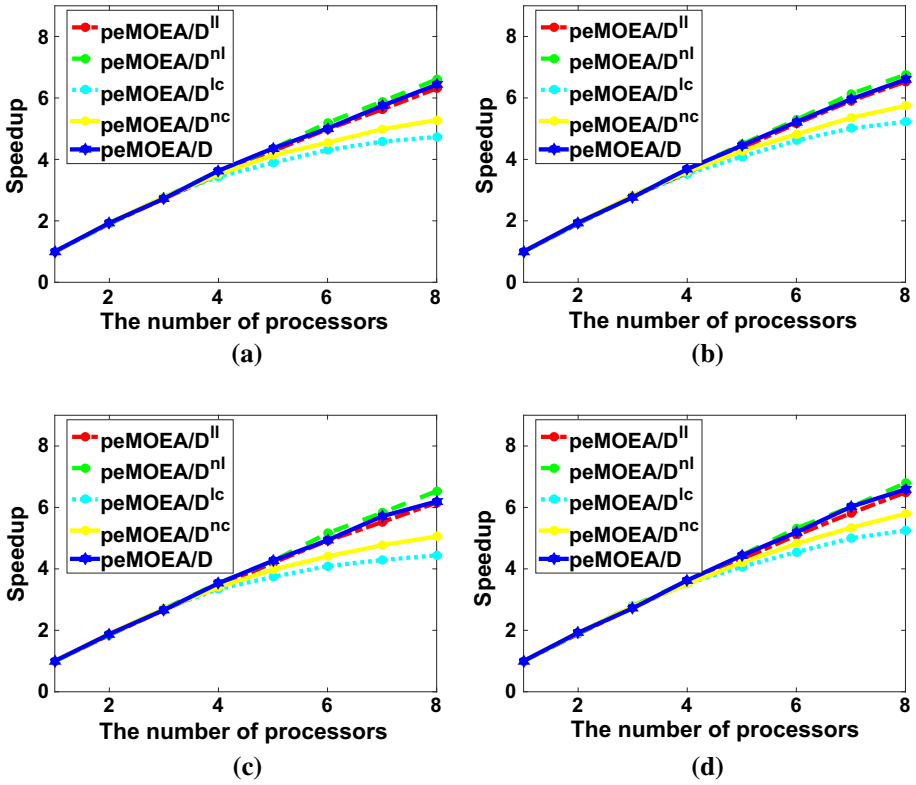

Fig. 13 Average speedup curves achieved by each variant of peMOEA/D with an increase in processors for three-objective **a** $DTLZ_1$, **b** $DTLZ_2$, **c** $DTLZ_3$ and **d** $DTLZ_4$

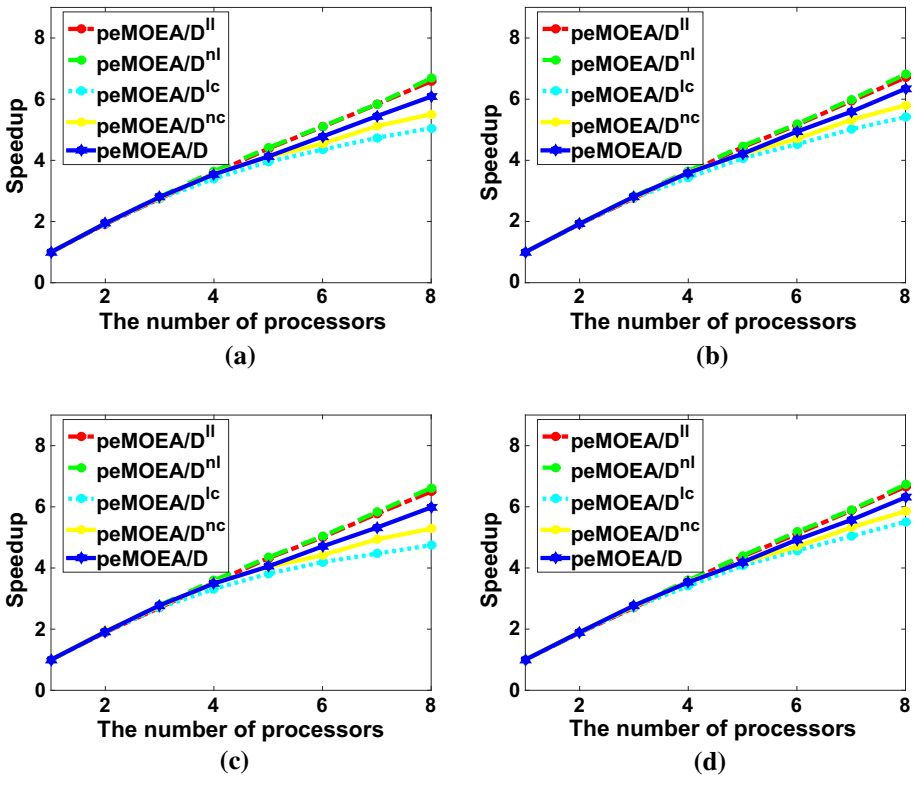

Fig. 14 Average speedup curves achieved by each variant of peMOEA/D with an increase in processors for five-objective **a** $DTLZ_1$, **b** $DTLZ_2$, **c** $DTLZ_3$ and **d** $DTLZ_4$

in processors for three- and five-objective $DTLZ_{1-4}$, respectively. It can be seen from these figures that the peMOEA/D^{nl} has a slightly higher speedup than the peMOEA/D while the peMOEA/D has a much higher speedup than the peMOEA/D^{nc} for three- and five-objective $DTLZ_{1-4}$. This suggests that migration traffic increases in turn from the

linear topology, the adaptive neighbor-based topology, to the complete graph topology. It is noticeable that speedup scalability of the peMOEA/D^{nc} deteriorates remarkably with an increase in processors as a result of the very heavy migration traffic caused by the complete graph topology. Furthermore, it can also be inferred from Figs. 13 and 14 that the peMOEA/D^{nl} has better speedup than the peMOEA/D^{ll} while the peMOEA/D^{nc} has better speedup than the peMOEA/D^{lc}. This is attributed to the fact that the nearest-neighbor partitioning approach results in fatter partitions and a lower probability of migrating elitist individuals. In conclusion, the peMOEA/D can achieve a satisfactory balance between convergence performance and speedup even for MOPs with three or more objectives owing to the partially evolved island model, the nearest-neighbor partitioning approach and the adaptive neighbor-based migration topology.

5 Conclusions

Although the overlapped partitioning MOEA/D, transplanted from the shared-memory parallel versions, can be executed on message-passing clusters, it is only suitable for bi-objective optimization due to the difficulty of maintaining overlapping individuals for MOPs with more than two objectives. The universal partially evolved parallel MOEA/D proposed in this paper effectively extends the opMOEA/D from a bi-objective space to higher dimensional objective spaces by using the partially evolved island model. This model not only eliminates the difficulty of maintaining overlapping individuals but also increases population diversity. Additionally, the peMOEA/D also adopts the nearest-neighbor partitioning approach, hybrid migration policy and adaptive neighbor-based topology. Fatter partitions generated by nearest-neighbor partitioning can reduce migration traffic of elitist individuals. Then, the hybrid migration policy helps subpopulations to cooperate more effectively across separate processors than either of the pure migration policies does. Finally, the adaptive neighbor-based topology keeps an excellent balance between convergence speed and migration traffic. Experimental results verify the satisfactory overall performance of the peMOEA/D in terms of both quality of solutions and speedup for MOPs with two or more objectives on message-passing clusters.

Our ongoing research is focused on applications of the peMOEA/D for various engineering problems including the layout of wireless sensor networks. Also, we will extend the peMOEA/D to some variants of MOEA/D for many-objective optimization problems. Finally, we plan to parallelize other decomposition-based MOEAs such as CAEA on message-passing clusters.

Acknowledgments This work was supported in part by the Natural Science Foundation of Guangdong Province, China (No. 2015A0303-13204), the Pearl River Science & Technology Nova Program of Guangzhou (No. 2014J2200052), the National Natural Science Foundation of China (Nos. 61203310 and 61503087), the China Scholarship Council (CSC) (Nos. 201406155076 and 201408440193) and the Fundamental Research Funds for the Central Universities, SCUT (No. 2013ZZ0048).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

Branke J, Schmeck H, Deb K, Reddy S (2004) Parallelizing multi-objective evolutionary algorithms: cone separation. In: Proceedings of the Congress on Evolutionary Computation (CEC-2004), vol 2. IEEE Press, pp 1952–1957

Cheshmehgaz HR, Desa MI, Wibowo A (2013) Effective local evolutionary searches distributed on an island model solving bi-objective optimization problems. Appl Intell 38(3):331–356

Clarke L, Glendinning I, Hempel R (1994) The MPI message passing interface standard. In: Programming environments for massively parallel distributed systems. Birkhäuser, Basel, pp 213–218

Coello C, Christiansen AD (2000) Multiobjective optimization of trusses using genetic algorithms. Comput Struct 75(6):647–660

Cutello V, Narzisi G, Nicosia G (2006) A multi-objective evolutionary approach to the protein structure prediction problem. J R Soc Interface 3(6):139–151

Deb K, Pratap A, Agarwal S, Meyarivan T (2002a) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

Deb K, Jain H (2014) An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints. IEEE Trans Evol Comput 18(4):577–601

Deb K, Thiele L, Laumanns M, Zitzler E (2002b) Scalable multi-objective optimization test problems. In: Proceedings of Congress on Evolutionary Computation. IEEE Press, pp 825–830

Deb K, Zope P, Jain A (2003) Distributed computing of pareto-optimal solutions with evolutionary algorithms. In: Evolutionary multi-criterion optimization. Springer, Berlin, pp 534–549

Durillo JJ, Zhang Q, Nebro AJ, Alba E (2011) Distribution of computational effort in parallel MOEA/D. In: Learning and intelligent optimization. Springer, Berlin, pp 488–502

Li H, Zhang Q (2009) Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II. IEEE Trans Evol Comput 13(2):284–302

Molina G, Alba E, Talbi EG (2008) Optimal sensor network layout using multi-objective metaheuristics. J Univ Comput Sci 14(15):2549–2565

Nebro AJ, Durillo JJ (2010) A study of the parallelization of the multi-objective metaheuristic MOEA/D. In: Learning and intelligent optimization. Springer, Berlin, pp 303–317

Streichert F, Ulmer H, Zell A (2005) Parallelization of multi-objective evolutionary algorithms using clustering algorithms. In: Evolutionary multi-criterion optimization. Springer, Berlin, pp 92–107

van Veldhuizen DA, Zydallis JB, Lamont GB (2003) Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Trans Evol Comput 7(2):144–173

Ying W, Xie Y, Xu X, Wu Y, Xu A, Wang Z (2015) An efficient and universal conical hypervolume evolutionary algorithm in three or

higher dimensional objective space. IEICE Trans Fundam Electron Commun Comput Sci E98-A(11):2330–2335

Ying W, Xu X (2012) Bi-objective optimal design of truss structure using normalized conical-area evolutionary algorithm. Int J Adv Comput Technol 4(15):162–171

Ying W, Xu X, Feng Y, Wu Y (2012) An efficient conical area evolutionary algorithm for bi-objective optimization. IEICE Trans Fundam Electron Commun Comput Sci E95-A(8):1420–1425

Zaharie D, Petcu D, Panica S (2008) A hierarchical approach in distributed evolutionary algorithms for multiobjective optimization. In: Large-scale scientific computing. Springer, Berlin, pp 516–523

Zhang Q, Li H (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11(6):712–731

Zhou A, Qu BY, Li H, Zhao SZ, Suganthan PN, Zhang Q (2011) Multiobjective evolutionary algorithms: a survey of the state of the art. Swarm Evol Comput 1(1):32–49

Zitzler E, Deb K, Thiele L (2000) Comparison of multiobjective evolutionary algorithms: empirical results. Evol Comput 8(2):173–195

Zitzler E, Laumanns M, Thiele L (2001) SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In: Evolutionary methods for design optimization and control with applications to industrial problems. International Center for Numerical Methods in Engineering, pp 95–100