Parallel transfer evolution algorithm

Yuanjun Laili, Lin Zhang, *and* Yun Li

Abstract*—Parallelization of an evolutionary algorithm takes the advantage of modular population division and information exchange among multiple processors. However, existing parallel evolutionary algorithms (PEAs) are rather *ad hoc* and lack a capability of adapting to the problem or platform environments. To accommodate a wider range of problems and to reduce algorithm design costs, this paper develops a parallel transfer evolution (PTE) scheme. This is based on the island-model of parallel algorithms and, for improving performance, transfers both the connections and the evolutionary operators from one sub-population pair to another adaptively. Needing no extra upper selection strategy, each sub-population becomes autonomously able to select evolutionary operators and local search operators as subroutines according to both the sub-population's own and the connected neighbor's ranking boards dynamically. The PTE scheme is tested on two typical combinatorial optimization problems in comparison with six existing *ad hoc* parallel evolutionary algorithms, and is also applied to a real-world case study in comparison with five typical parallel evolutionary algorithms. The tests show that the PTE scheme and the resultant PEA offer high flexibility in dealing with a wider range of combinatorial optimization problems without algorithmic modification or redesign. Both the topological transfer and the algorithmic transfer are seen applicable not only to combinatorial optimization problems, but also to continuous or non-permutated complex problems.[1]

***Index Terms*—Evolutionary computation, combinatorial optimization, parallel algorithm, topological design, algorithmic adaptation**

I. INTRODUCTION

REAL-world non-deterministic polynomial-time hard (NP-hard) optimization problems are becoming more complex to solve and are presenting more challenges to evolutionary algorithms (EAs). An EA mimics natural evolution with a population in generational iterations to search for feasible and optimal solutions to NP-hard problems [1]. In dealing with these problems, parallel evolutionary algorithms (PEAs) have become increasingly popular [2]. Intuitive parallelism is to divide the EA population into a number of sub-populations and map them onto multiple processors that work concurrently. It partitions the potential solution space, enhances global search for multi-peak problems, and gives more room to maneuver for algorithm hybridization. So far, PEAs have seen many successes in solving complex optimization problems [3,4].

In recent years, three main models of PEAs have been reported as design bases. These models are the master-slave model, the island model, and the diffusion model [5]. Meanwhile, hierarchical hybrid models combining one or more of these models have also been reported for certain special cases. Owing to the widespread use of multi-core computers and clusters, the island model [6–8] has become the most common, in which each sub-population evolves in an independent processor as an "island." The "islanders" interact periodically via individual migration, in accordance with a pre-defined topology. The resultant communication overheads are generally lower than in the master-slave and the diffusion models [9,10].

Owing to the structure of the island model, the migration policy and island topology are the most critical elements in determining the efficiency of the PEA.

The migration policy controls the migration frequency, the migration rate, the number of migrating individuals, the individual replacement rule, and the synchronization of the sub-populations [1,11]. Much research and many experiments have been reported on designing a migration policy in various scenarios, where certain offline schemes [3,12,13] and online strategies [14–16] are established not only to set the migration policy, but also to adaptively adjust key algorithmic parameters of the sub-populations during the runtime.

The island topology is also an important factor of the PEA in determining the neighbors of each sub-population for individual exchanges [17]. The most commonly used ones are the ring [18], mesh [19], full-mesh [43], and star topologies [20]. Generally, an island topology of a PEA is not easy to determine optimally, as communication objects of each sub-population are difficult to determine during the runtime. There are two major reasons for this. First, the correlation between the state of evolution and the topology is difficult to evaluate quantitatively. Second, the implementation means of a specific topology in a PEA is normally fixed. To deal with the above problems, studies on random topologies [21,22] and graph-based dynamic topologies [23,24] have been carried out. Those topologies are first randomly changed during the iteration and then are adapted to the problem structure [25]. However, the neighbors of each island need to be recalculated and broadcast according to the new structure in every iteration. This takes a long time, resulting in performance degradation on the parallel evolution. Today, the design of an efficient PEA with a low communication overhead remains a challenge.

One attempt to address this issue has been to tailor a PEA to the characteristics of the problem being tackled [26,27]. Another has been to assign multiple problem-dependent

Yuanjun Laili and Lin Zhang are with the School of Automation Science and Electrical Engineering, Beihang University, Beijing, 100191, China. (e-mail: llyj0721@gmail.com, zhanglin@buaa.edu.cn).

Yun Li is with School of Computer Science and Technology, Dongguan University of Technology, Dongguan, 523828, China, and with Faculty of Engineering, University of Strathclyde, Glasgow, G1 1XJ, U.K. (e-mail: Yun.Li@ieee.org).

Lin Zhang and Yun Li are the corresponding authors.

heuristics to a sub-population. A third approach has been to adapt the size of the sub-populations. Nevertheless, the migration policy and the topology are generally kept unchanged. The deficiency of these problem-specific PEAs is that once the characteristics of the problem change, the algorithm is hard to cope or adapt. This issue is partially addressed using memetic algorithms and hyper-heuristics with multiple EAs [28]. The most common design is to allocate a group of operators or memes (i.e., local search strategies) to different islands directly and let them interact with one another via individual migration [26,29–32]. However, it requires an extra upper-layer algorithm selection process to update individuals inside the sub-population, which will largely degrade the parallel efficiency as well. For a multiple-EA-based PEA, as its operators or upper-layer adaptation rules inside each sub-population are generally uniform, the performance of the PEA is lower than those of the underlying EAs if the islands are not well balanced. Therefore, research on self-adaptation of the island topology and dynamic selection of multiple EAs is imperative for PEAs.

To extend the current research, in this paper, we develop a parallel transfer evolution (PTE) scheme to structure a flexible PEA. At the topological level, the connection between sub-population pairs is transferred adaptively during each period of communication. With only one single connection, the communication overhead is maintained to be the minimum and the diversity of the sub-populations are also preserved. At the algorithmic level, superior operators can be transferred from a sub-population to its neighbor to enhance the search capability of the sub-populations. For applications, we focus on permutation-based combinatorial optimization, where multiple variables of the problem form a permutation such as in the case of a scheduling, assignment, or routing problem.

The remainder of the paper is structured as follows. In Section 2, we review the state-of-the-art PEAs. In Section 3, we provide a framework of the proposed PTE and detail its topological and algorithmic transfers. The PTE is then fully tested with the combination of several classical evolutionary operators on various benchmarks and on a real-world virtual channel scheduling problem found in communication systems, in Sections 4 and 5, respectively, giving comparisons with both traditional EAs and PEAs. Conclusions are drawn and potential future work is highlighted in Section 6.

II. State of the Art of PEAs

According to the number of evolutionary algorithms adopted in PEAs, existing research has focused mainly on two aspects to construct efficient PEAs progressively. These are *the design of PEA with a single EA* and *the adaptation of PEA with multiple EAs*.

A. The design of PEA with a single EA

The island model reveals that *migration policy* and *cooperative topology* are two crucial factors in the design of a PEA [2,33].

(1) Migration policy

For relatively simple problems, a linear or near linear speed-up can be achieved, owing to relatively even divisions of the population and the solution space [34]. For example, Alba [4] has summarized and classified performance evaluations on parallelization, and has given instances to show that a linear speed-up is possible in a PEA, although the population division reduces the search capability of each sub-population. Considering the diversity collapse phenomenon that results from the introduction of high-fitness individuals [35], Alba and Trova [36] studied the influence of random emigration on the population diversity and suggested when to use fitness-based or random emigration at different states of evolution. Qian *et al.* [37] further introduced parallel processors to generate new individuals for multi-objective optimization and adopted a merge strategy to accelerate the comparisons in updating a Pareto archive. With low communication overhead between the processors, this method was proved to be approximately linear both in theory and in practice.

(2) Cooperative topology

To obtain higher collaborative capability and search quality during parallel search, Cantú-Paz [11] introduced the concept of selection pressure and takeover time to evaluate the diversity and convergence of the entire population. Given the PEA topologies reported in [18–20], Matsumura *et al.* [17] compared them and concluded that the ring topology would simultaneously guarantee high population diversity and information diffusion with a single migration policy. However, Hijaze and Corne [38] and Wang *et al.* [39] applied these topologies to distinct cases and showed that the influence of each topology varies according to the context. In view of the performance limitations of a fixed topology, Giacobini *et al.* [40] investigated small-world graphs and scale-free graphs as new candidate topologies for the construction of the sub-populations. In addition, Li *et al.* [41] introduced β-graph-based network topologies and discussed their construction, complexity, and diversity. Liu *et al.* [42] have established an optimal r-regular graph topology for particle swarm optimization (PSO) and proved its efficiency both theoretically and practically.

(3) Adaptation in migration policy and cooperative topology

It has become clear that a uniform migration policy and topology cannot usually offer efficient collaboration among islands, as the states of the population at different search stages are different. Therefore, adaptive strategies in both migration policy and island topology are desirable.

For a migration policy, Noda *et al.* [14] provided a series of knowledge-based rules to guide the selection and replacement of migrants. Lardeux and Goëffon [15] proposed a dynamic strategy to control the migration probability based on a complete graph. Following these efforts, Yang and Tinos [43] provided an elite set, instead of a random or a high-fitness-based migration strategy, to determine which individuals to exchange. Further, Araujo and Merelo [16] applied entropy as a representation of diversity and tested various adaptive migration policies in accordance with the distance between the migrant and the target island. In addition, Zhan *et al.* [44] proposed a mean-fitness-rank-based approach to migrate individuals from poor-performing populations to better-performing populations, so as to maintain the diversity and a balanced search pace in the entire population. Their results provide comprehensive insight into the setting of a migration policy.

In addition, Whitacre *et al*. [45] attempted to make the topology co-evolve with the population by locality and interaction epistasis. Arnaldo *et al*. [25] placed an emphasis on the importance of the topology and hence attempted to match various island topologies to the problem structure by varying the topology at runtime for the first time. However, a drawback of this adaptation is that the topology-generating and co-evolving processes take a relatively long time and a relatively large amount of memory in a normal parallel programming environment, such as the message passing interface (MPI). To be specific, when the topology has changed, the algorithm needs to recalculate, store, and broadcast the communicating neighbors for each sub-population in every iteration. Hence, this method is inefficient and is seldom applied in practice. To reduce the communication overhead and improve the exchange dynamics between islands, Tao *et al*. [46] developed an adaptive pre-detection mechanism based on a full-mesh topology. This efficiently reduced the communication overhead in each iteration and simultaneously enhanced the search capability of the PEA developed therein.

B. *The adaptation of PEA with multiple EAs*

As a growing number of EAs have been developed in recent years, researchers have integrated multiple EAs in concurrent islands to realize parallel hybridization. The earliest memetic algorithms were developed based on this idea [47,48]. Subsequent representative parallelization work still follows this approach, and divides a population into islands in order to apply adaptive selection of memes to fine-grained individuals [29,49,50]. Although all of the algorithm (or meme) candidates act uniformly on each sub-population, tailored PEAs with a collaborative use of multiple EAs [30] are also developed for certain problems.

However, the migration policy and the topology are both static. Although multiple EAs are collected and the algorithm selection strategy for individuals is pre-designed [51–55], most of these schemes are unsuitable for a PEA for two reasons. First, with conventional parallelism the search capability of a PEA is not well maintained compared to its serial counterpart. Second, the strategies for both adjusting the action scope of an algorithm candidate and the sub-population size will result in load imbalance in different processors. With the increased time complexity of the PEA, the adjustment of algorithms among the sub-populations has not been addressed.

So far, studies on how to adapt a topology dynamically to implement flexible parallel search are very limited. Without a suitable algorithm adaptation mechanism for a PEA of multiple EAs, algorithms applied to specific problems will result in a lag in the search pace and reduce the algorithm efficiency. No matter how far the dynamics of the migration policy is explored, the search scope and diversity of the PEA are restricted, as the efficiency and flexibility of a PEA of multiple algorithms are far from fully exerted.

III. The Parallel Transfer Evolution Scheme

In this section, we first illustrate a framework of the PTE being proposed. Dynamic topological transfer and algorithmic transfer are elaborated following this framework. The evolutionary states used in the PTE are also analyzed.

A. *Main structure of the PTE*

The basic structure of the PTE is established as shown in Fig. 1. The execution process consists of three main steps: (1) sub-population evolution, (2) topological transfer, and (3) algorithmic transfer.

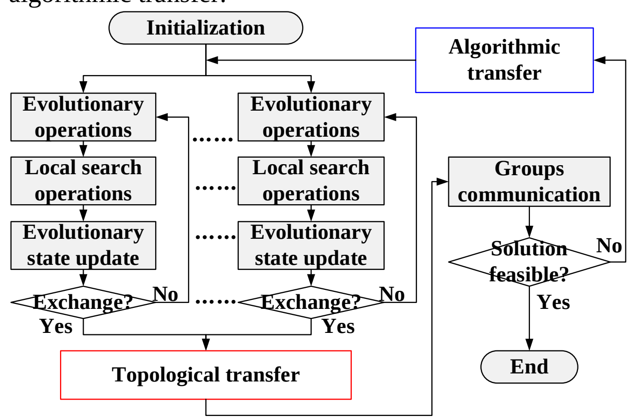


Fig. 1 Main structure of the PTE

(1) Sub-population evolution: This refers to the evolutionary operations combined with cdertain local search heuristics for producing a new sub-population in each generation. Inspired by memetic computing [49][50], the evolutionary operators are primarily applied for exploration, while the local search heuristics are adopted to exploit better solutions in a randomly located local area and therefore enhance the search capability.

(2) Topological transfer: To minimize the communication overhead in each period of exchange, we restrict the number of connections among the sub-populations to 1. The topological transfer then means to delete the existing connection and create a new one between another sub-population pair according to the updated evolutionary state.

(3) Algorithmic transfer: Instead of using an upper-layer algorithm selection mechanism on each sub-population, an algorithmic transfer is designed to immigrate superior evolutionary operator from the dynamic connected neighbor along with the individual to be migrated.

B. *Evolutionary states for communication control*

Despite parameter tuning in a single EA or the algorithm adjustment in multiple EAs, evolutionary states are of significant importance in both performance and evaluation control. The most commonly used states for a population include the best fitness ever found in the evolution process (*BF*), the number of generations for unchanged best fitness (*UN*), the variance of fitness values (*VF*), the convergence degree (*CD*), and the distance between two individuals (*D*). Assume that the best, the average and the worst fitness values of a sub-population for minimization problem in generation t is $f_{\min}(t)$, $\overline{f}(t)$ and $f_{\max}(t)$, $F_{\min}$ and $F_{\max}$ as the best and the worst fitness value that have been found ever by the specific sub-population. Then, the above states can be calculated as follow:

$$BF = \min f_{\min}(t)\,, \tag{1}$$

$$VF(t) = \sqrt{(f_{\max}(t) - \bar{f}(t))^2 + (\bar{f}(t) - f_{\min}(t))^2} \,/\, (F_{\max} - F_{\min} + \delta)\,, \tag{2}$$

$$CD = VF(t-1) / VF(t)\,, \tag{3}$$

In Eq. (2), δ is a small number that used to avoid the 'division by zero' error when $F_{\max} = F_{\min}$. Among the states, *VF* reflects the diversity of the current population, while *UN* and *CD* measure the convergence of the search process. The larger *VF* is, the higher the diversity.

Correspondingly, $CD \geq 1$ implies that the population is gradually converged, and $CD < 1$ implies an increase in diversity. According to *CD*, only the states of the recent two generations are reflected. As a supplement, *UN* offers another perspective on the convergence of the entire evolutionary process. Therefore, we define a generation convergence measure, *GM*, as the control state for the following step. It is calculated as

$$GM = (1 + UN) \cdot CD\,. \tag{4}$$

When *BF* is updated, $UN = 0$, and *GM* represents only the diversity of the current generation. Conversely, if $UN > 0$, then *GM* reflects a convergence degree of the entire iterative process.

It should be noted that there are many other metrics that can be used to evaluate the diversity of a population. Therefore, Eq. (2) can be replaced by other diversity formula to guide the following evolution.

For simplifying the evolutionary process and reducing the communication time, we set only one migrant and apply the above states to determine whether the best individual or a random one is to be sent out, as illustrated in Algorithm 1. The migration policy is that the immigrant is always introduced to replace the worst individual in the target sub-population.

Algorithm 1: Communication preset:

Step 1 **If** $rand() < 2/(1+e^{-GM}) - 1$
Step 2 Set the best individual as the migrant
Step 3 **Else**
Step 4 Randomly select an individual as the migrant

In the step of communication preset, *GM* is saturation-scaled by a sigmoid function within the interval (0,1). The smaller *GM* is, the greater the probability is in selecting a diverse individual.

C. Topological transfer

Among the typical topologies for PEAs, the ring topology has been seen as the most efficient, which can simultaneously guarantee a high population diversity and information diffusion with the same migration policy [17]. However, it appears that only the predominant migrant can produce useful impact on a specific sub-population. Other less competitive migrants introduced during periodic communication will be replaced quickly by the locally generated new individuals. Therefore, certain connections are unnecessary.

Since only the best migrant has a major impact on the search, this implies that the removal of other connections has almost no negative impact on the solution quality or convergence, and still produces a positive impact on acceleration owing to the decreased load in model communication. This means that we only need to migrate the predominant migrant in each communication period to one of the other groups. The migration destination can be randomly picked or designated using prior knowledge. Based on this analysis, a connection transfer mechanism is developed, as illustrated in Fig. 2.

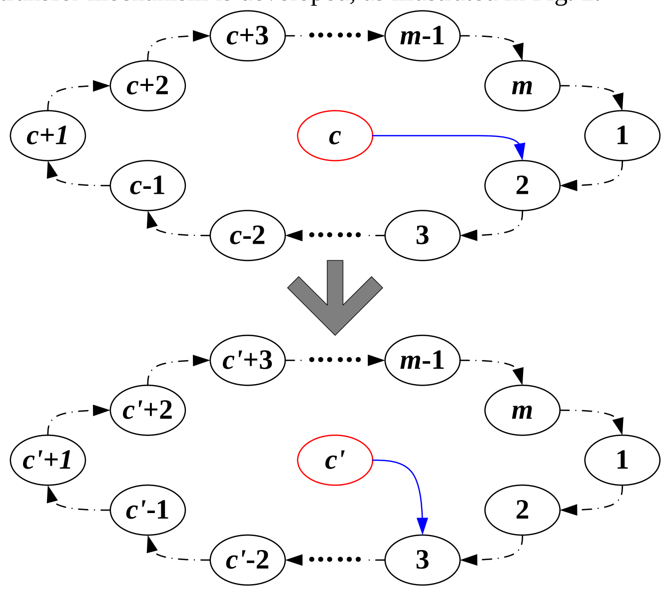


Fig. 2 Illustration of the connection transfer between sub-population pairs

Assume there are *m* sub-populations in total and the *c*-th sub-population holds the best individual obtained thus far till the current generation. Borrowing the virtue of ring topology, we pick the *c*-th sub-population as the sender at which to generate a single directed connection to one of the remaining sub-populations. The receiver is selected in a random ergodic manner by simply increase the serial number of the remaining sub-populations.

In the next period of communication, a new sender *c'* which holds the global best individual and a new receiver whose serial number is near the last one are selected. The connection will be transferred to the new pair as well. No matter how the position of the predominant sender changes, each sub-population can communicate with the global best sequentially during a certain period of time. The information propagation speed is exactly the same as in traditional ring topology, which is *m* times of communication at most. The additional computational load of finding the sender to which the most prominent individual belongs is taken by the root processor, i.e., the first processor. In finding the global best fitness value from *m* fitness values collected from all sub-populations, the additional computation complexity is only $O(m)$. More importantly, the communication load in each period is reduced to $O(m+n)$, where *n* refers to the dimension of the specific problem. That is, only *m* fitness values and a migrant with *n* dimensions are passed from the transferred connection in each communication period. This complexity is much lower than that in conventional topologies and other dynamic ones. To better understand the topological transfer process, its pseudo-code is shown in Algorithm 2.

Algorithm 2: Topological transfer

Step 1 Reduce the fitness value of the best individual f_x in each group x to the root processor;

Step 2 Find the sender $c' = \arg\min f_x, x \in [1, m]$;

Step 3 Set the receiver as $d' = (d + 1) \bmod m$;

Step 4 Send the global best individual from the new sender c' to the new receiver d';

Step 5 Replace the worst individual by the migrant in d'.

Here, d and d' represent the receiver of the last communication period and of the current period, respectively. It can be initially set as the root processor or a randomly selected one. In each period of communication, the receiver is changed one by one, as shown by the dashed lines in Fig. 2. Algorithm 2 is executed only in the root and the destination processors. Others will hand over their best individual to start the next generation independently. For maintaining synchronization, a communication check in each period of communication should be set.

D. Algorithmic transfer

Algorithmic transfer is established based on the above topological transfer. To record the performance of the under-layer evolutionary operators and find the superior one to be transferred, we introduce the tabu strategy presented by Burke *et al*. [54] for each sub-population. Assume that there are N_E evolutionary operators applied in the PTE scheme. In each sub-population i, we set a rank list $\mathbf{R_i} = \{R_{ik} \mid i \in [1, N], k \in [1, N_E]\}$ and a tabu list $\mathbf{T_i} = \{T_{ik} \mid i \in [1, N], k \in [1, N_E]\}$ to record the ranks and states of operators in the step of evolutionary state update, as shown in Algorithm 3. The operator with the highest rank in the sender will be passed accompanied by the emigrant individual to the receiver. The receiver can decide autonomously whether to apply the immigrant operator and individual or not, as demonstrated in Algorithm 4.

Algorithm 3: Rank record:

Step 1 **For** each sub-population i
Step 2 **If** BF_i is updated
Step 3 $R_{ik} = R_{ik} + 1$
Step 4 **Else**
Step 5 $R_{ik} = R_{ik} - 1$ and $T_{ik} = 1$
Step 6 **If** all operators are tabooed
Step 7 **For** k = 1 to N_E
Step 8 $T_{ik} = 0$
Step 9 Set E_i be the one with the highest rank $\max R_{ik}, k \in [1, N_E]$

Algorithm 4: Evolutionary operator configuration:

Step 1 **For** each sub-population i
Step 2 **If** the fitness value of $\mathbf{I_i}$ is better than BF_i
Step 3 Adopt O_i for the next generation
Step 4 **Else**
Step 5 Adopt E_i for the next generation

In the pseudo-code of Algorithms 3 and 4, BF_i represents the best fitness value of the sub-population i and $\mathbf{I_i}$ represents the immigrant of the sub-population i. Here, $\mathbf{I_i}$ is an n-dimensional vector to represent a solution. O_i represents the operator with the highest rank in the source sub-population that provided the immigrant $\mathbf{I_i}$.

Different from the strategy in [54], E_i is not directly used in the next generation, but sent to the neighboring group in the step of communication as O_i. With such a mechanism, the sub-populations are capable of exchanging good operators in each period and quickly eliminating weak operators for different sorts of problems. The selection of the evolutionary operators in this way is included in the communication. The transformation of only one index number will not increase the communication complexity, but will simplify the selection process and enhance the search efficiency significantly.

Following the information exchange, the evolution as designed will configure the operators according to both the local performance records and the incoming algorithm indices for the next generation. The pseudo-code is illustrated in Algorithm 4.

The time complexity of the rank record in steps 1–5 of Algorithm 3 is $O(1)$. From step 6 to step 9 of Algorithm 3, the computational complexity is $O(N_E)$ due to the parallel nature of sub-population. Additionally, Algorithm 3 uses two extra lists with length N_E to support the rank. Hence, both the total computational complexity and the space complexity of the algorithmic transfer including Algorithms 3 and 4 are $O(N_E)$.

To further improve the search efficiency of sub-population, local search heuristics are introduced in this paper. Local search is often used as a complementary component to enhance the exploitation of an EA. It is able to bring more neighborhood information for each individual to accelerate the evolutionary pace of the sub-populations. Without loss of generality, we assume that N_{LS} local search heuristics are collected after the evolutionary operation. Then, a random permutation-based mechanism as displayed in Algorithm 5 is brought to adjust several local heuristics for each individual in a sub-population.

Algorithm 5: Local search heuristic configuration:

Step 1 **For** each sub-population i
Step 2 **For** each individual j
Step 3 Get a random permutation $\mathbf{R_{perm}}$ from 1 to N_{LS}
Step 4 $T = T_0,\ k = 1$
Step 5 **While** $T > T_{\text{end}}$
Step 6 $LS_{ij} = R_{\text{perm}}[k]$
Step 7 Apply the No. LS_{ij} operator to individual j
Step 8 $\Delta = f_{\text{old}}(i, j) - f_{\text{new}}(i, j)$
Step 9 **If** $\Delta > 0 \,\|\, rand() < \exp(\Delta / T)$
Step 10 $\mathbf{I_{ij,old}} = \mathbf{I_{ij,new}}$
Step 11 $T = T \cdot T_{\text{decay}}$
Step 12 **If** $k > N_{LS}$
Step 13 Regenerate $\mathbf{R_{perm}}$ and set $k = 1$
Step 14 $k = k + 1$

In the above pseudo-code, $\mathbf{R_{perm}}$ represents a randomly generated permutation and T, T_0, and T_{decay} represent the current annealing temperature, the initial temperature, and the decay rate, respectively. Further, $f_{\text{old}}(i,j)$ and $f_{\text{new}}(i,j)$ represent the old and new fitness values, respectively, of individual j in sub-population i before and after the local search. $\mathbf{I_{ij,old}}$ and $\mathbf{I_{ij,new}}$ represent the old individual j and the new individual j in sub-population i before and after the local search. In order to promote a short local search time, a small initial temperature $T_0 = 10$ and fast decay rate $T_{\text{decay}} = 0.9$ are set in this paper.

Observably, this is a classical random permutation selection strategy combined with an annealing rule to control step length. There are two reasons for applying such a random strategy. For combinatorial optimization, the shape of the solution space is usually irregular and even unknown. When we reach a point in the solution space, we actually do not know which kinds of local search heuristics should be used to search its near range without problem-dependent information. More importantly, a local search heuristic that is suitable for one point in searching its neighborhood may not adaptable for another point because they are located in entirely different landscapes. Therefore, passing the local search heuristic that performs well in a sub-population to its neighbor as well as the evolutionary operators seems meaningless.

Following the two-layer operations, i.e., the evolutionary operation and local search operation, the stopping criterion is set as either the theoretical optimum is reached or as the maximum number of generations is reached, or *GM* is larger than a predetermined threshold $GM_{\max}$.

In general, the time complexity of an evolutionary operator is dynamically varied with different problems. Let g_{E_i} and g_{LS_j} be the complexity of the i-th evolutionary operator and the j-th local search heuristic, respectively, the complexity of the evolutionary operation be $\max g_{NS_i}$, and the complexity of the local search operation be $\max g_{E_i}$. Furthermore, we set the size of the sub-populations as N_{sub} uniformly. The complexity of the topological communication is $O(m+n)$ and the evolution is $O(N_E + \max g_{E_i}) + O(N_{sub} N_{LS} + \max g_{LS_i})$. Hence, the PTE is highly dependent on its operator candidates employed in generating new populations.

It should be noted that local search is not a necessary part in the framework of the PTE if the candidate evolutionary operators are capable of operating a balanced exploration and exploitation. Likewise, the local search heuristics can also be replaced by a group of problem-related rules. In short, the PTE is more likely a parallel pattern that can be used to integrate multiple evolutionary operators and local search heuristics in a collaborative form, and that can generate more extendable and fast hybrid algorithms.

IV. Experimental Tests on Two Combinatorial Optimization Problems

In this section, we comprehensively test the performance of the PTE on a generic combinatorial optimization problem, the job-shop scheduling problem (JSP), which is often seen in the manufacturing industry [66]. We also test it on a second generic combinatorial optimization problem, the quadratic assignment problem (QAP) [72].

The JSP is a problem to search for an effective dispatch sequence with a minimal machining makespan *C*. Given *n* jobs $J_1, J_2, \cdots, J_n$ of varying sizes, each job consists of a certain number of operations, which should be performed by *m* identical machines. Assume that $O(i,j)$ is the operation of job *j* processed by machine *i*, p_{ij} is the processing time of $O(i,j)$, C_{ij} is the completion time of $O(i,j)$, and M_j is the set of machines by which job *j* is processed.

The objective is

$$\text{Min } C_{\max} \tag{5}$$

s.t.

$$C_{\max} \ge C_{ij},\ C_{ij} - p_{ij} \ge C_{kl},\ C_{ij} - p_{ij} \ge 0,$$
$$C_{ij} - p_{ij} \ge C_{kj} \text{ or } C_{kj} - p_{kj} \ge C_{ij},\ i,k \in M^j, i \ne k,$$
$$C_{ij} - p_{ij} \ge C_{il} \text{ or } C_{il} - p_{il} \ge C_{ij},\ j \ne l.$$

The QAP is a combinational optimization problem in which *n* facilities need to be duly located among *n* locations. Given a set of facilities *P* and locations *L*, $c(p_1, p_2)$ represents the commodities of a certain flow between facilities p_1 and p_2, and $d(l_1, l_2)$ represents the distance between locations l_1 and l_2. Considering a problem of size *N*, we define a bijective function $f: P \to L$.

The objective is

$$Min \sum_{p_1, p_2 \in P} c(p_1, p_2) \cdot d(f(p_1), f(p_2)). \tag{6}$$

A. Experimental settings

To solve a generic, permutation-based combinatorial optimization problem, we adopt an integer coding scheme to represent solution phenotypes in evolution. The PTE is capable of being configured with existing EAs, and 12 such EAs used in the scheme are listed in Table 1, with explanations of acronyms used hereafter. The learning operator of PSO, CMPSO, and 5 types of DE algorithms are replaced with the "swap operator" and "swap sequence" recommended in [64] to ensure that the new real-coded individual is a complete permutation sequence. For the same reason, a two-point swapping mechanism is applied as the basic operation to HS, ILS, and VNS.

TABLE 1 Evolutionary algorithm examples used in the PTE

Abbreviation	Evolutionary algorithm
GA	Genetic algorithm [56] with swap sequence and swap operator
NGA	Genetic algorithm with niched strategy [57]
PSO	Particle swarm optimization [58]
CMPSO	Particle swarm optimization with Cauchy mutation [59]
DE1	Differential evolution with rand/1 mutation [60]
ILS	Iterative local search [61]
DE2	Differential evolution with best/1 mutation [60]
DE3	Differential evolution with rand/2 mutation [60]
HS	Harmony search [62]
DE4	Differential evolution with best/2 mutation [60]
DE5	Differential evolution with target-to-best/1 mutation [60]
VNS	Variable neighborhood search [63]

The PTE is designed to be able to utilize the 9 local search heuristics reported in [65], which are listed in Table 2. We assume that the length of the block in the local search heuristics is no more than n and is also randomly generated within the interval [2, n] on every call. The population size in all of these experiments is set to 40 and the maximum number of generations is set to $1000N$, where N is the number of variables in the test functions. In comparison, the migration period is set to 100.

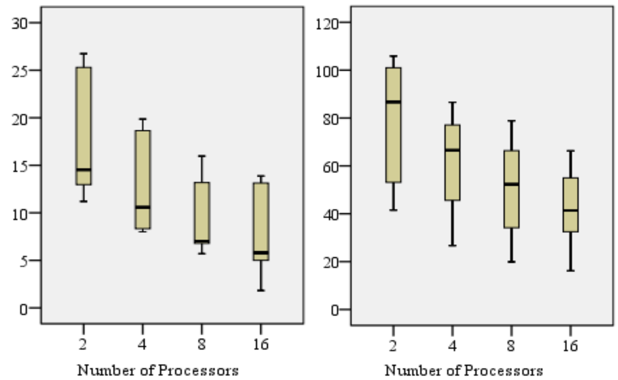
TABLE 2 Local search heuristics [65] used in the PTE

Abbreviation	Strategies
Swap	select two points and swap them
Pre-insert	select a point and insert it in the head of the sequence
Pos-insert	select a point and insert it in the tail of the sequence
Swap-block	Exchange two selected blocks in a sequence
Pre-block-insert	Insert a selected block in the head of the sequence
Pos-block-insert	Insert a selected block in the tail of the sequence
Swap-max/min	Swap a point with the maximal or the minimal value
Rand-circle	Exchange two adjacent points with a given probability from the beginning to end successively
Inverse	Select a piece of sequence and inverse it

All the simulations are coded in C++ and tested in a MAC OS X environment with a clang and MPI compiler. The hardware configuration is based on a 2.3-GHz Intel Core i7 CPU with 8 GB of 1.6-GHz DDR3 RAM. There are four cores in total. All of the tests are run 20 times. The best fitness values, the average fitness values, and the search times for each problem are recorded and compared with the best-known solutions (BKSs).

B. Results and discussions in solving the JSP and the QAP

The results of the PTE for the JSP benchmark instances (i.e. LA21-LA40) are summarized in the boxplots as shown in Fig. 3. For the 5 simple instances (LA23, LA31-LA33, and LA35) shown in Fig. 3(a), the speed-up from 2 processors to 4 processors is significant, although the speed-up from 4 processors to 16 processors is less significant. For the rest 15 harder instances, the speed-up is well observed in Figs. 3(b). Overall, with the increase of the processor number, the decision times of the PTE are nearly linearly reduced.



(a) (b)

Fig. 3 Boxplots corresponding to the CPU times (s) of the PTE on two groups of JSP instances, (a) PTE on LA23, LA31–LA33, and LA35, and (b) PTE on LA21, LA22, LA24–LA30, LA34, and LA36–LA40.

To compare the efficiency of the topological transfer and algorithmic transfer, the PEA with a ring topology and a random algorithm selection mechanism in each sub-population is tested and termed a Ring-PEA in this paper. The Ring-PEA is a typical parallel scheme without a topology configuration or an algorithm configuration. Similarly, the PEA with only topological transfer and random selection of the evolutionary operators in sub-population is also tested and termed a PTE/AT.

TABLE 3 COMPARISON OF THE RING-PEA, PTE/AT, AND PTE IN SOLVING THE JSP (20 RUNS)

Best	**Ring-PEA**				**PTE/AT**				**PTE**			
Time (s)	**2**	**4**	**8**	**16**	**2**	**4**	**8**	**16**	**2**	**4**	**8**	**16**
LA21	1113	1111	1074	1097	1087	1079	1073	1070	**1046**	**1046**	**1046**	**1046**
	95.1284	62.0936	43.2733	50.6899	82.2437	53.0961	18.3006	33.9262	**50.8246**	**42.8212**	**34.1372**	**32.5141**
LA22	962	945	942	953	941	932	935	939	935	**927**	**927**	932
	64.3132	60.0469	42.4670	45.8086	58.3960	51.5039	41.4189	28.1890	53.1203	**45.5661**	**40.2871**	37.3018
LA23	1038	1044	**1032**	1051	**1032**	**1032**	**1032**	**1032**	**1032**	**1032**	**1032**	**1032**
	50.0005	64.1751	**17.5194**	52.2266	**45.4322**	**17.0080**	**9.3262**	**10.8048**	**11.2068**	**8.0023**	**6.8349**	**5.7106**
LA24	998	984	1006	1002	967	991	991	985	939	**935**	**935**	940
	72.3252	55.6767	46.1206	42.1305	64.7975	49.3416	55.2058	38.5012	59.7715	**45.7829**	**40.0118**	34.5910
LA25	1028	1053	1029	991	1025	1022	986	1004	984	**977**	**977**	986
	78.3462	64.1038	50.9584	101.963	68.9161	52.0715	54.2061	39.2465	66.3077	**56.0303**	**48.3889**	38.8948
LA26	1282	1239	1250	**1218**	1261	**1218**	1221	1247	**1218**	**1218**	**1218**	**1218**
	136.123	91.8122	82.8408	**68.8742**	90.2576	**81.9801**	69.3567	55.4596	**96.7305**	**86.5698**	**68.1117**	**55.7823**
LA27	1312	1342	1286	1286	1313	1296	1281	1292	1256	**1249**	**1249**	1256
	207.838	122.413	103.432	88.7586	89.4123	55.5565	53.9323	30.0097	103.781	**85.1423**	**78.8039**	66.2894
LA28	1300	1289	1276	1285	1289	1233	1260	1260	1232	**1216**	1222	1235
	133.143	98.4433	74.4165	66.7367	90.9272	79.263	62.9837	29.4939	105.883	**86.4273**	76.6387	63.9851
LA29	1274	1261	1233	1250	1246	1240	1233	1245	1216	**1210**	**1210**	1215
	175.0565	138.516	135.700	92.6267	82.0519	62.2127	32.5237	25.9899	86.6730	**67.9818**	**58.7412**	54.9975
LA30	1408	1391	1392	1356	1391	1387	**1355**	1367	**1355**	**1355**	**1355**	**1355**
	87.6503	83.8414	80.4635	56.0052	62.1359	36.0310	**37.3242**	25.1080	**53.6943**	**45.8109**	**30.7131**	**23.4347**
LA31	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**
	46.4500	**39.1031**	**23.4566**	**26.8770**	**22.4226**	**16.2061**	**15.9824**	**9.2351**	**12.9699**	**10.5546**	**7.0003**	**5.8920**
LA32	**1850**	**1850**	**1850**	**1850**	**1850**	**1850**	**1850**	**1850**	**1850**	**1850**	**1850**	**1850**
	104.114	**45.9913**	**28.9017**	**49.4104**	**38.2363**	**28.9859**	**18.4508**	**12.8825**	**14.3852**	**10.6065**	**8.2136**	**5.3321**
LA33	**1719**	**1719**	**1719**	**1719**	**1719**	**1719**	**1719**	**1719**	**1719**	**1719**	**1719**	**1719**
	89.3159	**41.9318**	**33.8317**	**34.6119**	**21.9266**	**18.9948**	**8.1248**	**6.3302**	**12.0076**	**9.4833**	**6.7910**	**4.9963**
LA34	**1721**	**1721**	**1721**	**1721**	**1721**	**1721**	**1721**	**1721**	**1721**	**1721**	**1721**	**1721**
	175.595	**165.773**	**74.0150**	**85.3784**	**54.8783**	**57.0182**	**40.6377**	**29.9956**	**48.4446**	**27.3473**	**25.6408**	**22.8219**
LA35	**1888**	**1888**	**1888**	**1888**	**1888**	**1888**	**1888**	**1888**	**1888**	**1888**	**1888**	**1888**
	56.2926	**34.5302**	**35.9220**	**30.3115**	**19.5839**	**20.5997**	**16.8295**	**15.7469**	**15.0536**	**12.2713**	**7.0639**	**6.0064**
LA36	1378	1349	1319	1298	1315	1300	1296	1316	1296	**1268**	**1268**	1296
	139.8962	81.2775	70.2488	64.4265	95.9613	58.4591	45.9586	33.6278	100.983	**66.5589**	**52.2995**	41.3735
LA37	1496	1473	1463	1469	1493	1471	1471	1470	1434	**1422**	**1422**	1434
	103.956	92.1454	64.533	51.6995	65.4058	45.6660	46.9057	40.3616	94.4580	**67.6901**	**60.1195**	49.8878

LA38	1311 150.018	1297 93.6981	1297 80.0434	1287 80.0374	1312 93.2303	1289 70.0681	1268 65.9753	1270 43.0126	1237 103.881	1237 77.1148	**1222** **68.3370**	1237 61.8856
LA39	1358 126.519	1311 98.8735	1287 80.7876	1311 67.3212	1267 71.869	1266 44.8561	1256 38.2662	1279 26.4701	1252 98.7793	**1248** **76.5589**	1252 66.3664	1257 53.6549
LA40	1288 105.828	1297 82.0641	1297 91.2737	1287 80.6748	1280 93.2163	1258 71.5151	1259 72.5371	1255 60.2887	1244 102.378	**1233** **80.9734**	1244 65.7741	1244 50.8519
***P* of W-test**	**0.000**	**0.0 00**	**0.000**	**0.0 00**	**0.001**	**0.015**	**0.009**	**0.017**	-	-	-	-

In addition, the local search heuristics with a random selection strategy (shown in Algorithm 5) are applied in both the Ring-PEA and the PTE/AT to make sure that they are tested under the same conditions as the PTE. The best results and average search times of the Ring-PEA, PTE/AT, and PTE on 20 JSP instances (i.e., LA21–LA40) are shown in Table 3. The boldface in the table indicates that the best known solution is found within a specific time.

TABLE 5 COMPARISON OF WILCOXON-TEST RESULTS OF THE PTE RELATIVE TO 6 AD HOC EAS IN SOLVING THE JSP

PTE vs	Nowicki *et al*. (1996) [66]	Goncalves *et al*. (2005) [67]	Aiex *et al.* (2003) [68]	Binato *et al*. (2002) [69]	Sha *et al.* (2006) [70]
	0.004	0.069	0.415	0	0.003
	0.042	0.563	0.219	0	0.028
	0.021	0.476	0.261	0	0.018
	0.003	0.065	0.374	0	0.003

The performance of the Ring-PEA is seen as the worst. Only the 6 simplest instances (LA26 and LA31–LA35) are well solved with the best known solution (i.e. BKS). When the topological transfer is implemented, the PTE/AT performs much better than the original Ring-PEA. As observed in Table 3, 9 instances are solved with BKS by the PTE/AT. Its search times with two parallel processors are decreased to 19.5839 and 95.9613 s. As the number of processors continues to increase, the CPU times are further reduced to 6.3302 s at most.

When the algorithmic transfer is implemented, the performance of the PTE is further enhanced. 16 instances are well solved by the PTE within the BKS. As the number of processors increases further, the CPU times are reduced to 66.2894 s at least and 4.9963 s at most. To examine the differences between the other PEAs and the PTE, pair-wise Wilcoxon-tests (abbreviated as W-tests) are carried out at a significant level of $\alpha = 0.05$. The statistical test results of each kind of PEA are compared in a pairwise manner with those obtained by the PTE with the same processor number and are listed in the last two rows of Table 3. With 95% confidence, the PTE performs better than the Ring-PEA and PTE/AT.

TABLE 4 SOLUTION CONSISTENCY OF THE PTE WITH PARALLEL PROCESSORS COMPARED TO 6 AD HOC EAS IN SOLVING THE JSP (20 RUNS)

	BKS	Nowicki *et al.* (1996) [66]	Goncalves *et al.* (2005) [67]	Aiex *et al.* (2003) [68]	Binato *et al*. (2002) [69]	Sha *et al.* (2006) [70]	Zhang *et al*. (2013) [71]	PTE			
								2	**4**	**8**	**16**
LA21	1046	1047	1046	1057	1091	1046	1049	**1046**	**1046**	**1046**	**1046**
LA22	927	**927**	935	927	960	927	-	935	**927**	**927**	932
LA23	1032	**1032**	1032	1032	1032	1032	-	1032	**1032**	**1032**	1032
LA24	935	939	953	954	978	935	940	939	**935**	**935**	940
LA25	977	977	986	984	1028	977	982	984	**977**	**977**	986
LA26	1218	**1218**	1218	1218	1271	1218	-	1218	**1218**	**1218**	1218
LA27	1235	1236	1256	1269	1320	1235	1243	1256	1249	1249	1256
LA28	1216	**1216**	1232	1225	1293	1216	-	1232	**1216**	1222	1235
LA29	1157	1160	1196	1203	1293	1163	1180	1216	1210	1210	1215
LA30	1355	**1355**	1355	1355	1368	1355	-	**1355**	**1355**	**1355**	**1355**
LA31	1784	**1784**	1784	1784	1784	1784	-	**1784**	**1784**	**1784**	**1784**
LA32	1850	**1850**	1850	1850	1850	1850	-	**1850**	**1850**	**1850**	**1850**
LA33	1719	**1719**	1719	1719	1719	1719	-	**1719**	**1719**	**1719**	**1719**
LA34	1721	**1721**	1721	1721	1721	1721	-	**1721**	**1721**	**1721**	**1721**
LA35	1888	**1888**	1888	1888	1888	1888	-	**1888**	**1888**	**1888**	**1888**
LA36	1268	**1268**	1279	1287	1334	1268	1274	1296	**1268**	**1268**	1296
LA37	1397	1407	1408	1410	1457	1397	1408	1434	1422	1422	1434
LA38	1196	1196	1219	1218	1267	1196	1196	1237	1237	1222	1237
LA39	1233	1233	1246	1248	1290	1233	1238	1252	1248	1252	1257
LA40	1222	**1229**	1241	1244	1259	1224	1233	1244	1233	1244	1244

Moreover, the performance of the PTE is further analyzed and compared with 6 EAs developed elsewhere specifically for the JSP. The experimental results, pairwise Wilcoxon-tests carried out on the 6 *ad hoc* EAs and PTE are shown in Tables 4–6, respectively. When the processors are set to two, the PTE does not perform well. However, as the number of processors increases, the solution quality is substantially enhanced. When the number of processors is 4 or 8, the PTE is better than the EAs proposed by Nowicki *et al*. [65] and by Binato *et al*. [68], with 95% confidence ($\alpha = 0.05$), similar to the EAs proposed by Goncalves *et al*. [66] and by Aiex *et al*. [67], although not as good as the EA proposed by Sha *et al*. [69]. With only a group of basic operators, the search time of the PTE is 10 times shorter than the others, as shown in Table 6.

TABLE 6 TIMES (IN S) TAKEN BY THE PTE COMPARED TO 6 AD HOC EAS IN SOLVING THE JSP

Problem	Nowicki *et al*. (1996) [66]	Goncalves *et al*. (2005) [67]	Aiex *et al*. (2003) [68]	Binato *et al*. (2002) [69]	Sha *et al.* (2006) [70]	Zhang *et al*. (2013) [71]	PTE			
							2	**4**	**8**	**16**
LA21-25	-	602	-	-	295	-	49.75	39.4011	33.7274	29.0959
LA26-30	-	1303	-	-	579	-	90.2606	74.8469	62.6559	52.9083
LA31-35	-	3691	-	-	1462	-	20.5722	14.0526	10.9419	9.0097
LA36-40	-	1920	-	-	471	-	100.0959	73.7792	62.5793	51.5307

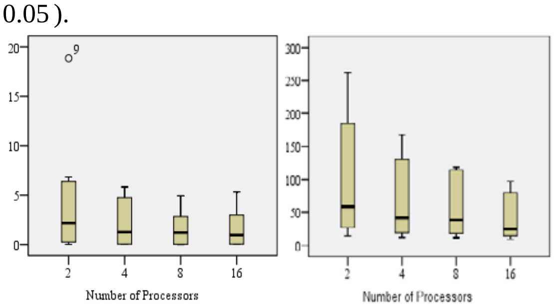
When the number of processors increases, the solution quality in small-scale cases is fully maintained with a reduced CPU time. For large-scale cases, the solution quality first rises and then falls. In particular, a sub-population of fewer individuals does not necessarily lead to an accuracy loss while improving the search performance. For all instances, when the processor number is increased from 2 to 8, the search precision of the PTE improves constantly. When the number of processors is further increased to 16, the search precision bounces back, but is still well maintained.

In summary, the PTE performs the best when the processor number is 4 or 8 on a quad-core PC. As the number of islands increases, the transfer scheme automatically come into effect. The more processors are allowed, the more dynamic the search procedure becomes. However, with a fixed population size (i.e., 40) in the experiments, only two individuals are left in each island when the processor number increases to 16, and, hence, the search capability of each sub-population is reduced.

TABLE 7 COMPARISON OF RING-PEA, PTE/AT, AND THE PTE IN SOLVING THE QAP (20 RUNS)

	Ring-PEA				**PTE/AT**				**PTE**			
	2	**4**	**8**	**16**	**2**	**4**	**8**	**16**	**2**	**4**	**8**	**16**
tai20a	727364	723784	718186	726538	710926	709874	710878	714344	**703428**	**703428**	**703428**	**703428**
	24.6509	9.0214	6.2282	4.3471	16.4254	7.1748	4.4127	2.7793	**4.7438**	**4.2874**	**2.8354**	**2.9927**
tai30a	1897942	1874605	1873775	1882734	1865652	1855449	1849745	1845521	1829732	**1821056**	1823521	1825967
	22.0127	11.4434	13.8688	8.7986	18.1725	14.2919	11.7346	5.9932	6.3922	**4.7536**	4.5312	5.3304
tai40a	3250393	3249185	3229880	3238796	3245172	3444240	3221913	3231247	3245517	**3206459**	3218664	3229871
	30.4285	18.2784	14.7997	11.9469	16.1815	14.7448	12.7171	7.9684	6.8449	**5.7087**	4.9339	5.0891
tai50a	5105137	5096802	5092834	5121268	5094182	5091133	5092607	5104653	5054377	5043459	**5043166**	5046897
	82.4618	53.8743	45.2217	29.7112	72.1527	45.272	38.2201	21.7837	18.8471	12.3823	**11.8429**	14.4364
tai60a	7509121	7471640	7443125	7490329	7451748	7418543	7431658	7450792	7401193	**7387519**	7389182	7390278
	126.237	81.0541	76.2499	42.2345	101.234	57.4519	44.9596	20.0815	27.1386	**23.8925**	22.3276	19.2474
tai80a	13993058	13970413	13821857	13801963	13945345	13947566	13804047	13804269	13708571	**13614458**	13632845	13636457
	215.733	103.851	88.2193	79.1816	175.521	77.2492	65.7479	35.1161	89.0165	**59.3624**	54.4672	29.7950
tai100a	21866427	21724890	21681753	21658299	21684735	21622438	21652433	21630649	21623739	**21497643**	21571329	21595571
	317.507	218.811	223.478	193.671	215.974	156.848	111.855	88.1446	184.399	**130.389**	114.457	97.1206
W-test	**0.000**	**0.000**	**0.000**	**0.000**	**0.001**	**0.000**	**0.000**	**0.001**	-	-	-	-

The experimental results of the Ring-PEA, the PTE/AT, and the PTE on 7 hard instances of the QAP are shown in Table 7. Here the best solutions obtained the twelve experiments and the p-values which are less than 0.05 in the statistical tests are shown in bold. With only topological transfer, the PTE/AT performs much better than the Ring-PEA in both solution quality and search time. After implementing the algorithmic transfer, the search time and capability of the PTE are further enhanced. This is mainly because suitable operators can be broadcasted to all sub-populations faster. Better operators are able to make the evolution more efficient, accelerate the convergence process, and shorten the search time. Further, pairwise W-tests are also carried out and illustrated in the last two rows of Table 7. The statistical differences between the two-PEAs and the PTE are significant, with 95% confidence ($\alpha = 0.05$).



(a) (b)

Fig. 4 Boxplots corresponding to the CPU times (s) of the PTE on two groups of QAP instances, (a) tai12a, tai12b, tai15a, tai15b, tai17a, tai20a, tai30a, tai40a, and tai50a, and (b) tai64c, tai60a, tai80a, tai100a, wil50, and wil100.

Because most of the algorithms existing in the literature for the QAP are evaluated by using the average percent deviation (APD) as a metric [72–75], we also present a list of the APD results obtained by the PTE in Table 8.

As shown in Table 8, the optimum solutions of the small instances, tai20a, tai30a, tai40a, and tai50a, are obtained within 15 s. For the remaining larger-scale instances, sub-optimal solutions are obtained in no more than 130.389 s when the processor number is two and at most 97.1206 s when the processor number reaches 16. The errors between these results and the theoretical solution are no more than 0.034.

TABLE 8 AVERAGE PERCENT DEVIATIONS OF THE QAP BY THE PTE

QAP	**BKS**	**PTE**			
		2	4	8	16
Tai20a	703428	0.000	0.000	0.000	0.000
Tai30a	1818146	0.006	0.002	0.003	0.004
Tai40a	3139370	0.034	0.021	0.025	0.029
Tai50a	4938796	0.023	0.021	0.021	0.022
Tai60b	7205962	0.027	0.025	0.025	0.026
Tai80b	13499184	0.015	0.008	0.010	0.010
Tai100a	21052466	0.027	0.021	0.025	0.026

Viewing the experimental results for both the JSP and QAP, it can be seen that the PTE maintains a good performance for different problems. It requires neither reconfiguration of algorithms and the parallel connections nor prior information or domain knowledge of the problem. In summary, the PTE has offered a high speed, scalable speed-up, good precision, and great robustness of optimization in solving the above two different complex problems.

V. VIRTUAL CHANNEL SCHEDULING CASE STUDY

In this section, we apply the PTE to a practical engineering problem, the virtual channel scheduling (VCS) problem, as a case study in communication systems.

A. *Virtual channel scheduling problem*

The VCS problem is detailed and modeled in [76], which is a complex NP-hard problem. It refers to scheduling various sorts of virtual channel (VC) services in different time slots. The target of VCS is to maintain stable and fast transmission by maximizing throughput and minimizing delay time, jitter, and loss packet rate. Different characteristics of VC services and multiple quality of service (QoS) requirements make it much more complex than a generic scheduling problem. Assume that $n_i^{(k)}$ is the decision variable to denote whether VC$_i$ is scheduled in the kth time slot, M the number of time slots, l the number of VCs, and C the data transmission rate for a downlink. The objective function and constraints of VCS can be represented as follows:

$$\text{Max} \sum_{i=1}^{l-1} w_1^i \cdot Throughput_i(n_i) - \sum_{i=0}^{l-1} w_2^i \cdot Loss_i(n_i) \quad (7)$$

s.t.

$$\sum_{i=1}^{l} n_i^{(k)} = 1\,,\ C \cdot \sum_{k=1}^{M} n_i^{(k)} / M = B_0\,,\ Jitter_i(n_i) \le Jit_i\,,\ \text{if } i = 2,3$$

$$Delay_i(n_i) \le Del_i\,,\ \text{if } i = 0,2,3\ B_i \le C \cdot \sum_{k=1}^{M} n_i^{(k)} / M \le 1.4B_i\,,\ \text{if } i = 1,2\,.$$

TABLE 9 COMPARISON BETWEEN VARIOUS IMPLEMENTATIONS OF THE PEAS IN THE VCS APPLICATION

500		Ring-PEA	DRing-PEA	Mesh-PEA	DMesh-PEA	Fullmesh-PEA	PTE/AT	PTE
2	**Best**	113.765	116.165	115.176	117.311	114.499	114.929	117.356
	Avg	110.998	114.659	114.385	115.181	111.445	113.891	117.169
	Time (s)	27.599	28.1989	18.5247	24.9852	16.7536	14.5672	17.1838
4	**Best**	116.896	117.213	115.033	117.439	117.054	117.173	117.489
	Avg	115.910	116.606	113.49	115.828	115.374	115.73	117.373
	Time (*s*)	14.2695	16.8463	15.718	14.0113	14.9735	13.8726	10.8517
8	**Best**	116.52	117.114	115.927	116.461	117.252	117.41	117.588
	Avg	114.891	115.571	115.473	114.07	116.79	116.244	117.335
	Time (*s*)	12.7416	15.1329	13.83	8.5304	13.7529	8.4920	8.1612
16	**Best**	115.67	117.331	116.303	117.331	116.758	116.798	117.390
	Avg	115.552	115.769	114.682	115.788	115.849	115.636	117.234
	Time (*s*)	10.1887	17.9764	12.4829	8.9190	11.2249	11.2322	7.6419
1000		**Ring-PEA**	**DRing-PEA**	**Mesh-PEA**	**DMesh-PEA**	**Fullmesh-PEA**	**PTE/AT**	**PTE**
2	**Best**	117.853	115.878	117.963	117.811	117.143	118.626	120.277
	Avg	115.851	114.556	116.282	115.948	116.301	116.45	119.557
	Time (*s*)	37.0390	41.5085	35.3106	44.2789	53.0654	37.2425	43.2352
4	**Best**	118.655	119.397	118.903	118.189	119.199	118.309	120.413
	Avg	117.747	117.109	117.205	117.786	118.025	117.568	119.949
	Time (*s*)	29.3923	33.6408	26.9667	32.6701	21.7468	21.6442	20.5146
8	**Best**	118.685	118.470	119.192	118.29	119.409	118.767	120.524
	Avg	116.728	115.505	117.319	116.853	117.888	117.644	120.005
	Time (*s*)	22.4295	22.7115	16.3728	38.5605	23.7505	21.6442	16.8335
16	**Best**	117.643	117.768	117.359	117.815	118.08	118.69	120.193
	Avg	115.876	115.152	116.291	114.628	116.921	116.472	119.831
	Time (*s*)	29.454	28.3161	21.1185	26.5072	25.6297	23.0913	15.9788
1500		**Ring-PEA**	**DRing-PEA**	**Mesh-PEA**	**DMesh-PEA**	**Fullmesh-PEA**	**PTE/AT**	**PTE**
2	**Best**	116.135	113.722	114.754	118.324	118.163	116.878	119.524
	Avg	113.765	112.785	111.872	115.128	116.351	114.208	118.591
	Time (*s*)	90.006	85.7479	92.603	91.8577	91.272	71.7323	46.1519
4	**Best**	119.695	118.918	117.930	119.508	119.728	118.838	119.927
	Avg	118.637	117.301	116.944	113.879	118.641	117.692	119.781
	Time (*s*)	50.3336	65.836	77.8648	62.0776	65.644	42.2182	34.6408
8	**Best**	116.53	115.976	118.695	118.850	119.53	117.418	120.117
	Avg	106.474	114.591	116.359	116.641	117.962	116.587	119.627
	Time (*s*)	43.6652	58.8007	49.0954	55.4981	60.7339	27.6428	25.8103
16	**Best**	117.57	116.003	116.700	117.821	118.805	116.263	119.679
	Avg	115.585	115.497	112.063	115.205	115.155	115.776	119.306
	Time (*s*)	43.2569	40.1273	58.226	46.417	62.9013	34.0557	22.4259

In the above formulation, Del_i, Jit_i, and B_i are the maximum delay time, maximum jitter, and maximum bandwidth of VC$_i$, respectively. The delay ($Delay_i(n_i)$), jitter ($Jitter_i(n_i)$), throughput ($Throughput_i(n_i)$), loss packet rate ($Loss_i(n_i)$), and weights (w_1^i and w_2^i) are calculated as in [76]. The variables $n_i^{(k)}$ can be mapped as a permutation within the interval [0,l]. Each number in the permutation denotes the virtual channel to be scheduled in a current time slot. All the initial settings of VCS in our experiments are the same as in [76]. For testing the problem in different scales, 500, 1000, and 1500 time slots are set as three different cases. The more time slots that are used, the smaller they are, and hence the higher the precision that is expected.

B. *Algorithmic settings*

In this application, the PTE is tested with 5 typical topologies, i.e., single-sided ring, double-sided ring, single-sided mesh, double-sided mesh, and full mesh topologies. For uniformity, the local search operators in accordance with Algorithm 5 are applied in all of the PEAs. Without loss of generality, the population size of the EAs is set to 40 and the maximum number of generations is set to $1000N$. Then, a group of tests based on three scales of VCS are carried out to compare different parallel methods to the PTE. To analyze the influences of algorithmic parameters on VCS optimization, the migration period and the threshold $GM_{\max}$ are studied in this subsection.

C. *PTE performance and topology comparison*

The results of the 5 classical topologies compared to the PTE/AT and the PTE are shown in Tables 9 and 10. With different problem scales and characteristics, the performance of these classical topologies change as well, and these changes are usually inconsistent. The PTE always finds better solutions and takes much shorter search time. It offers a significant speed-up without precision loss while the processor number increases, as shown in Fig. 5. When the number of processors increases from 2 to 4, the execution time of the PTE is reduced by almost half. When the number of processors continues to increase, the time is slightly reduced due to sharing of computing cores among different processors.

TABLE 10 COMPARISON OF THE W-TEST AMONG VARIOUS IMPLEMENTATIONS OF THE PEAS FOR VCS

PTE vs		Ring-PEA	DRing-PEA	Mesh-PEA	DMesh-PEA	Fullmesh-PEA	PTE/AT
Time (s)	W-test	0	0	0	0	0	0
	W-test	0.008	0.003	0.019	0.002	0.003	0.034

The best average times for the cases of 500, 1000, and 1500 time slots are 7.6419, 15.9788, and 22.4259 s, respectively. Compared to the other 6 PEAs under the same conditions, the PTE is up to 3 times faster.

Without changing any part of the algorithm, the PTE is seen as capable of performing well not only in the benchmark tests, but also in solving a practical problem in diverse circumstances. Offering a shortened search time in both small-and large-scale problems, the search capability of the PTE is well maintained at a high level.

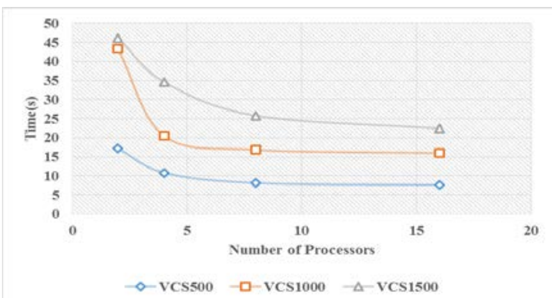


Fig. 5 Speed-up of the PTE on solving the VCS with three scales

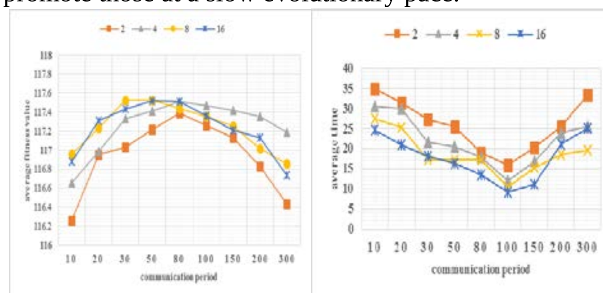
D. *Parameter tuning of the PTE in VCS*

In this subsection, we discuss the performance of the PTE with varied communication periods and different $GM_{\max}$ values. First, we vary the communication period from 10 to 300 and test the search capability of, and times taken by, the PTE with 2, 4, 8, and 16 processors. The average fitness values and solution times are presented in Fig. 6.

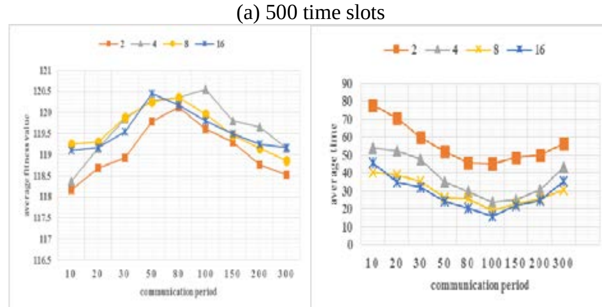
Regarding the average fitness values, we notice that the solution quality ascends during periods 10 to 100 and descends as the period continues to increase. With the increase in processor number, the peak values move higher, i.e., the PEA with more sub-populations requires a shorter communication period. In contrast, a long communication period is better for the PEA with fewer sub-populations to maintain balanced search states. When the processor number of the PTE is altered, the performance trend appears to be the same as that in the three benchmark tests.

On the search time, we observe from Fig. 6 that the performance of the PTE reaches the best level when the communication period is set as 100. The performance trends in all three cases for VCS are similar. When the communication period is lower than 200, its search time in each case decreases with an increasing number of processors. When this number grows, exchanges are delayed. As a result, the convergence speed lowers, especially for 16 processors, where there are only two individuals left in each sub-population.

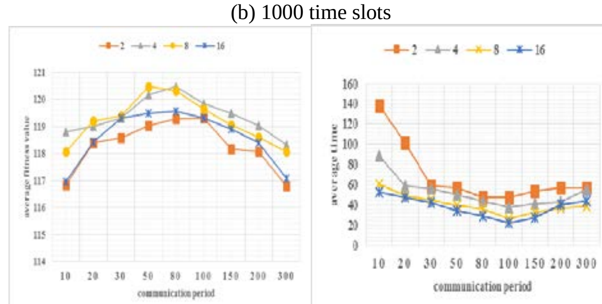
In view of both search quality and search time, we observe that when the communication period is in the range [50,100], the performance of the PTE is in a very good state. While the processor number continues to increase, a shorter period may help accelerate the exchanges among the sub-populations so as to promote those at a slow evolutionary pace.



(a) 500 time slots



(b) 1000 time slots

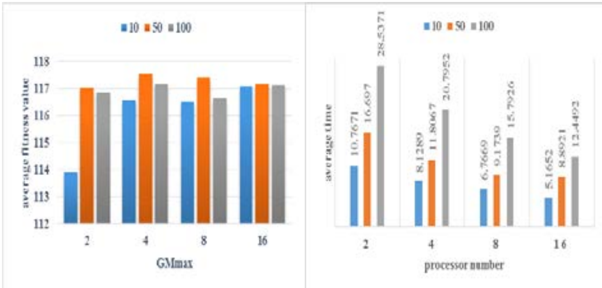


(c) 1500 time slots

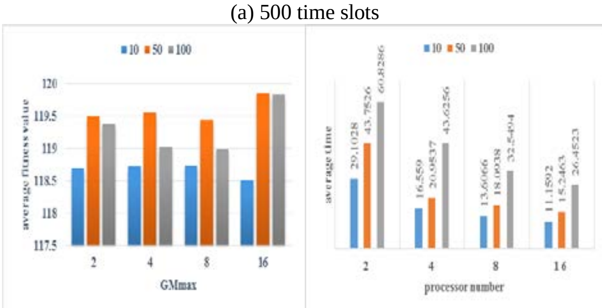
Fig. 6 Average results and search times of the PTE with different communication periods for three VCS cases

Subsequently, we fix the communication period to 100 and change $GM_{\max}$ from 10 to 100 to test the search capability of, and the times taken by, the PTE with 2, 4, 8, and 16 processors. The average fitness values and average solution times for the three instances are illustrated in Fig. 7.

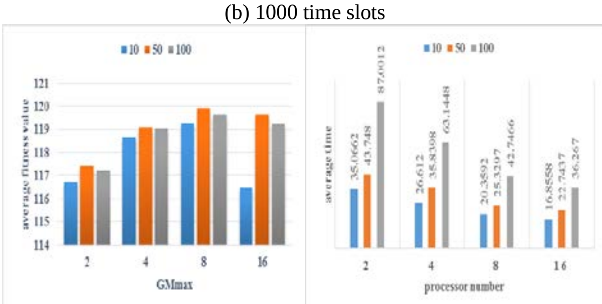
Because $GM_{\max}$ is one of the stopping criteria, the smaller $GM_{\max}$ is, the quicker the program terminates. From Fig. 7, we observe that when $GM_{\max}$ is 10, the PTE performs the worst. According to Algorithm 1, a small $GM_{\max}$ will bring about many randomly selected migrants and make the evolution process terminate earlier. If $GM_{\max}$ is set to 50, the search performance is markedly improved and acceptable. Nevertheless, when it is further increased, the performance slightly degrades. When GM_{max} is set to 100, for example, almost no random migrant is selected for communication, and the current best solution will be transformed again and again in an early stage. Comparing this to the case GM_{max}=50, the search time becomes prolonged, and a frequent transformation of the current best solution is reduced to some extent.



(a) 500 time slots



(b) 1000 time slots



(c) 1500 time slots

Fig. 7 Solutions and search times of the PTE with different GM_{max} values in solving three VCS cases

It is noted that for all of the PEAs in the above experiments that, as the number of processors increases from 8 to 16, the search time of every PEA does not decrease distinctly. This is mainly because of the limited population size (which is set as 40 in all of the above experiments) and the limited hardware resource adopted in the experiments. 8 or 16 processors are compressed into 4 cores to execute with resource preemption so as to slightly slow down the processing speed that the algorithm should have. However, this does not mean that the performance of the proposed method is limited to 8 processors in parallel. It is fully extendable to larger population size with more hardware cores. Theoretically, the increase of individuals in a population will only increase the search scope so as to improve the solution quality. If more hardware resources can be adopted for more sub-populations, the diversity of the entire evolutionary process will be largely enhanced. Therefore, more cores and more population will only benefit the performance of a PEA, but not limit its evolutionary efficiency.

In short, the PTE is designed as a scheme to expand the existing evolutionary operators in a highly flexible parallel way and make them faster and more adaptable in solving different sorts of combinatorial optimization problems.

VI. Conclusions

The focus of this paper has been to establish a parallel transfer scheme to structure parallel evolutionary algorithms flexibly to handle a wider range of real-world optimization problems. Using a group of classical evolutionary operators and local search heuristics, we have demonstrated both the communication connection and the evolutionary operator in PTE are able to transfer through sub-population pairs and thus to improve PEA performance. The PTE enables efficient collaboration among sub-populations with minimal communication.

To test the performance of the PTE in solving combinatorial optimization problems, comprehensive experiments have been carried out on the generic JSP and QAP problems, as well as by applying PTE to a practical VCS problem. In most cases, the PTE has outperformed other EAs and PEAs, especially in search speed and quality. Furthermore, the speed-up on the parallelism is approximately linear, while degradation of solution accuracy is avoided.

Both the topological transfer and algorithmic transfer are applicable not only to combinatorial optimization problems, but also to continuous or non-permutated complex problems. Classical evolutionary operators and local search heuristics can both be replaced by other subroutines directly to configure and form a new algorithm. Therefore, we are motivated to apply the PTE scheme to more practical domain following extended comparisons between the parallel hyper-heuristic based evolutionary algorithms and the PTE. It is expected that this scheme is capable of solving not only single-objective, but also multi-objective, optimization problems for engineering practices and in changing environments.

Acknowledgement

This work is partially supported by the "Young Talent Lift Project" of China Association for Science and Technology, the Natural Science Foundation of China under Grant 61374199, the Beijing Natural Science Foundation under Grant 4142031 and the 863 program project in China under Grant 2013AA041302.

References

[1] E Alba, M Tomassini. Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation, 2002, 6(5): 443-462.

[2] E Alba. Parallel metaheuristics: a new class of algorithms. John Wiley & Sons, 2005.

[3] A Lopez Jaimes, C A Coello Coello. MRMOGA: a new parallel multi-objective evolutionary algorithm based on the use of multiple resolutions. Concurrency and Computation-Practice & Experience, 2007, 19(4): 397-441.

[4] E Alba. Parallel evolutionary algorithms can achieve super-linear performance. Information Processing Letters, 2002, 82(1): 7-13.

[5] D A Van Veldhuizen, J B Zydallis, G B Lamont. Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computations, 2003, 7(2): 144-173.

[6] H R Cheshmehgaz, M I Desa, A Wibowo. Effective local evolutionary searches distributed on an island model solving bi-objective optimization problems. Applied Intelligence, 2013, 38(3): 331-356.

[7] L delaOssa, J A Gamez, J A Puerta. Initial approaches to the application of islands-based parallel EDAs in continuous domains. Journal of Parallel and Distributed Computing, 2006, 66(8): 991-1001.
[8] S C Lin. Coarse-grain parallel genetic algorithms: categorization and new approach. The 6th IEEE Symposium on Parallel and Distributed Processing, 1994: 28-37.
[9] X Y Zhang, J Zhang, Y J Gong, Z H Zhan, W N Chen, Y Li. Kuhn-Munkres parallel genetic algorithm for the set cover problem and its application to large-scale wireless sensor networks. IEEE Transactions on Evolutionary Computation, 2016, 20(5): 695-710.
[10] K E Parsopoulos. Parallel cooperative micro-particle swarm optimization: a master-slave model. Applied Soft Computing, 2012, 12(11): 3552-3579.
[11] E Cantú-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms. Journal of heuristics, 2001, 7(4): 311-334.
[12] S Skolicki, K De Jong. The influence of migration sizes and intervals on island models. Proceedings of the 2005 conference on Genetic and evolutionary computation. ACM, 2005: 1295-1302.
[13] J Lassig, D Sudholt. Design and analysis of migration in parallel evolutionary algorithm. Soft Computing, 2013, 17(7): 1121-1144.
[14] E Noda, A L V Coelho, I L M Ricarte I L M, A Yamakami, A A Freitas. Devising adaptive migration policies for cooperative distributed genetic algorithms. IEEE International Conference on Systems, Man and Cybernetics, 2002, 6: 6-pp.
[15] F Lardeux, A Goëffon. A dynamic island-based genetic algorithms framework. Simulated Evolution and Learning. Springer Berlin Heidelberg, 2010: 156-165.
[16] L Araujo, J Julian Merelo. Diversity through multiculturality: assessing migrant choice policies in an island model. IEEE Transactions on Evolutionary Computation, 2011, 15(4): 456-469.
[17] T Matsumura, M Nakamura, J Okech, K Onaga. A parallel and distributed genetic algorithm on loosely-coupled multiprocessor system. IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences, 1998, 81(4): 540-546.
[18] M L M Beckers, E P P A Derks, W J Melssen, L M C Buydens. Using genetic algorithms for conformational analysis of biomacromolecules. Computers & Chemistry, 1996, 20(4): 449-457.
[19] Y Fukuyama, H D Chiang. A parallel genetic algorithm for generation expansion planning. IEEE Transactions on Power Systems, 1996, 11(2): 955-961.
[20] H Miyagi, T Tengan, S Mohanmed, M Nakamura. Migration effects on tree topology of parallel evolutionary computation. IEEE Region 10 Conference on TENCON, 2010: 1601-1606.
[21] F M Defersha, M Chen. A parallel genetic algorithm for dynamic cell formation in cellular manufacturing systems. International Journal of Production Research, 2008, 46(22): 6389-6413.
[22] F M Defersha, M Chen. A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. International Journal of Advanced Manufacturing Technology, 2010, 49(1-4): 263-279.
[23] L Li, J M Garibaldi, N Krasnogor. Automated self-assembly programming paradigm: The impact of network topology. International Journal of Intelligent Systems, 2009, 24(7): 793-817.
[24] J M Whitacre, R A Sarker, Q T Pham. The self-organization of interaction networks for nature-inspired optimization. IEEE Transactions on Evolutionary Computation, 2008, 12(2): 220-230.
[25] I Arnaldo, I Contreras, D Millán-Ruiz, J I Hidalgo, N Krasnogor. Matching island topologies to problem structure in parallel evolutionary algorithms. Soft Computing, 2013, 17(7): 1209-1225.
[26] C Segura, E Segredo, C Leon. Scalability and robustness of parallel hyperheuristics applied to a multiobjectivised frequency assignment problem. Soft Computing, 2013, 17: 1077-1093.
[27] J Jin, T G Crainic, A A Løkketangen. A cooperative parallel metaheuristic for the capacitated vehicle routing problem. Computers & Operations Research, 2014, 44: 33-41.
[28] E G Talbi. A taxonomy of hybrid metaheuristics. Journal of Heuristics, 2002, 8: 541-564.
[29] J Tang, M H Lim, Y S Ong. Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems. Soft Computing, 2007, 11: 873-888.
[30] W Deng, R Chen, J Gao, Y Song, J Xu. A novel parallel hybrid intelligence optimization algorithm for a function approximation problem. Computers & Mathematics with Applications, 2012, 63(1): 325-336.
[31] F Tao, Y J Laili, Y Liu, Y Feng, Q Wang, L Zhang, L Xu. Concept, principle and application of dynamic configuration for intelligent algorithms. IEEE Systems Journal, 2014, 8(1): 28-42.
[32] F Tao, L Zhang, Y J Laili. Configurable Intelligent Optimization Algorithms: Theory and Practice in Manufacturing. Springer, 2014.
[33] E Alba, A J Nebro, J M Troya. Heterogeneous computing and parallel genetic algorithms. Journal of Parallel and Distributed Computing, 2002, 62(9): 1362-1385.
[34] T C Belding. The distributed genetic algorithm revisited. Proceedings of the 6th International Conference on Genetic Algorithms, 1995: 113-121.
[35] J Denzinger, J Kidney. Improving migration by diversity. Proceedings of the Congress on Evolutionary Computation, 2003, 1: 700-707.
[36] E Alba, J M Troya. Influence of the migration policy in parallel distributed GAs with structured and panmictic populations. Applied Intelligence, 2000, 12(3): 163-181.
[37] C Qian, J C Shi, Y Yu, K Tang, Z H Zhou. Parallel pareto optimization for subset selection. Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16), New York, 2016: 1939-1945.
[38] M Hijaze, D Corne. An investigation of topologies and migration schemes for asynchronous distributed evolutionary algorithms. Proceedings of the World Congress on Nature and Biologically Inspired Computing, IEEE, 2009.
[39] G Wang, D Wu, K Szeto. Quasi-parallel genetic algorithms with different communication topologies. The IEEE Congress on Evolutionary Computation (CEC), 2007: 721-727.
[40] M Giacobini, M Preuss, M Tomassini. Effects of scale-free and small-world topologies on binary coded self-adaptive cea. Lecture Notes in Computer Science, 2006, 3906: 86-98.
[41] L Li, J M Garibaldi, N Krasnogor. Automated self-assembly programming paradigm: the impact of network topology. International Journal of Intelligent Systems, 2009, 24(7): 793-817.
[42] Q Liu, W Wei, H Yuan, Z H Zhan, Y Li. Topology selection for particle swarm optimization. Information Sciences, 2016, 363: 154-173.
[43] S Yang, R Tinos. A hybrid immigrants scheme for genetic algorithms in dynamic environments. International Journal of Automation & Computing, 2007, 4(3): 243-254.
[44] Z H Zhan, Y Lis, J Zhang. Cloudde: a heterogeneous differential evolution algorithm and its cloud version. IEEE Transactions on Parallel and Distributed Systems, 2016.
[45] J Whitacre, R Sarker, Q Pham. The self-organization of interaction networks for nature-inspired optimization. IEEE Transactions on Evolutionary Computation, 2008, 12(2): 220-230.
[46] F Tao, Y Laili, L Xu, L Zhang. FC-PACO-RM: a parallel method for service composition optimal-selection in cloud manufacturing system. IEEE Transactions on Industrial Informatics, 2013, 9(4): 2023-2033.
[47] H Muhlenbein. Evolutionary in time and space: the parallel genetic algorithm. Foundations of Genetic Algorithms, 1991: 316-337.
[48] P Moscato, M G Norman. A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. Parallel Computing and Transputer Applications, 1992: 177-186.
[49] J Tang, M H Lim, Y S Ong. Parallel memetic algorithm with selective local search for large scale quadratic assignment problems. International Journal of Innovative Computing Information and Control, 2006, 2(6): 1399-1416.
[50] J Tang, M H Lim, Y S Ong. Adaptation for parallel memetic algorithm based on population entropy. The 8th Annual Genetic and Evolutionary Computation Conference (GECCO 2006), ACM SIGEVO, 2006, 1-2: 575-582.
[51] J A Vrugt, B A Robinson, J M Hyman. Self-adaptive multimethod search for global optimization in real-parameter spaces. IEEE Transactions on Evolutionary Computation, 2009, 13(2): 243-259.
[52] D Hadka, P Reed. Borg: an auto-adaptive many-objective evolutionary computing framework. Evolutionary Computation, 2013, 21(2): 231-259.
[53] J Grobloer, A P Engelbrecht, G Kendall, V S S Yadavalli. Alternative hyper-heuristic strategies for multi-method global optimization. IEEE Congress on Evolutionary Computation, 2010: 1-8.
[54] E K Burke, G Kendall, E Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics, 2003, 9(6): 451-470.
[55] C Qian, K Tang, Z H Zhou. Selection hyper-heuristics can provably be helpful in evolutionary multi-objective optimization. In: Proceedings of the 14th International Conference on Parallel Problem Solving from Nature (PPSN'16), Edinburgh, Scotland, 2016: 835-846.
[56] J Holland. Adaptation in natural and artificial systems. The University of Michigan Press, 1975.
[57] K De Jong, An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan, Ann Arbor, 1975.

[58] J Kennedy, R C Eberhart. Particle swarm optimization. IEEE International Conference on Neural Networks, 1995.
[59] H Wang, Y Liu, S Zeng. A hybrid particle swarm algorithm with Cauchy mutation. IEEE Swarm Intelligence Symposium (SIS 2007), 2007: 356-360.
[60] W Gong, Z Cai, C X Ling, H Li. Enhanced differential evolution with adaptive strategies for numerical optimization. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, 2011, 41(2): 397-413.
[61] H R Lourenco, O C Martin, T Stutzle. Iterated local search. arXiv preprint math/0102188, 2001.
[62] Z W Geem, J H Kim, G V Loganathan. A new heuristic optimization algorithm: harmony search. Simulation, 2001, 76(2): 60-68.
[63] N Mladenovic, P Hansen. Variable neighborhood search. Computers & Operations Research, 1997, 24(11): 1097-1100.
[64] K P Wang, L Huang, C G Zhou, W Pang. Particle swarm optimization for travelling salesman problem. Proceedings of The 2nd International Conference on Machine Learning and Cybernetics, 2003.
[65] P C Ma, F Tao, Y L Liu, L Zhang, H X Lu, Z Ding. A hybrid particle swarm optimization and simulated annealing algorithm for job shop scheduling. IEEE International Conference on Automation Science and Engineering (CASE), 2014.
[66] E Nowicki, C Smutnicki. A fast taboo search algorithm for the job-shop problem. Management Science, 1996, 42(6): 797-813.
[67] J F Goncalves, J J de Magalhaes, M G C Resende. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research, 2005, 167: 77-95.
[68] R M Aiex, S Binato, M G C Resende. Parallel GRASP with path-relinking for job shop scheduling. Parallel Computing, 2003, 29: 393-430.
[69] S Binato, W J Hery, D M Loewenstern, M G C Resende. A GRASP for job shop scheduling. Essays and Surveys in Metaheuristics, Springer, 2002: 59-79.
[70] D Y Sha, C Y Hsu. A hybrid particle swarm optimization for job shop scheduling problem. Computers & Industrial Engineering, 2006, 51: 791-808.
[71] R Zhang, S Song, C Wu. A hybrid artificial bee colony algorithm for the job shop scheduling problem. International Journal of Production Economics, 2013, 141: 167-178.
[72] T James, C Rego, F Clover. Multistart tabu search and diversification strategies for the quadratic assignment problem. IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans, 2009, 39(3): 579-596.
[73] Y Marinakis, A Migdalas. A hybrid genetic-GRASP algorithm using lagrangean relaxation for the traveling salesman problem. Journal of Combinatorial Optimization, 2005, 10: 311-326.
[74] J Yang, C Wu, H P Lee, Y Liang. Solving traveling salesman problems using generalized chromosome genetic algorithm. Progress in Natural Science, 2008, 18: 887-892.
[75] T James, C Rego, F Glover. A cooperative parallel tabu search algorithm for the quadratic assignment problem. European Journal of Operational Research, 2009, 195: 810-826.
[76] Y Zhu, P Wan, Y Chen, F Tao, L Zhang. Modeling and Solution for Virtual Channel Scheduling for Downlink Business. Asia Simulation Conference (AsiaSim 2014), Japan, 2014.