A Distributed Cooperative Coevolutionary Algorithm for Multiobjective Optimization

K. C. Tan, Y. J. Yang and T. H. Lee
Department of Electrical and Computer Engineering
National University of Singapore
10 Kent ridge Crescent Singapore 19260

Abstract- Evolutionary techniques have become one of the most powerful tools for solving multiobjective optimization (MOO) problems. However the computational cost involved in terms of time and hardware often become surprisingly burdensome as the size and complexity of the problem increases. This paper proposes a distributed cooperative coevolutionary algorithm (DCCEA), which evolves multiple solutions in the form of cooperative subpopulations and exploits the inherent parallelism by sharing the computational workload among computers over the network. Through its multiple features such as archiving, dynamic sharing and extending operator, solutions of DCCEA are not only pushed to the true Pareto front but also well distributed. Simulation results show that DCCEA has a very competitive performance and reduces the runtime effectively.

1 Introduction

Due to their inherent parallelism and their capability to evolve a family of solutions concurrently in a single run, evolutionary algorithms have been recognized to be well suited for multiobjective optimization problems. Since Schaffer's work, evolutionary techniques for multiobjective optimization have been gaining significant attention from researchers in various fields and new MOEAs (Corne et al, 2000; Deb et al, 2002; Knowles and Corne, 2000; Tan et al, 2001, 2003; Zitzler et al, 2001) continue to be developed. The main differences among existing multiobjective evolutionary algorithms are in selective fitness assignment and diversity maintenance.

Recent advances in evolutionary algorithms indicate that the introduction of ecological models and the use of coevolutionary architectures are effective ways to broaden the use of traditional evolutionary algorithms (Rosin and Belew, 1997; Potter and De Jong, 2000). Coevolution can be classified into competitive coevolution and cooperative coevolution. While competitive coevolution tries to get more competitive individuals through evolution, the goal of cooperative coevolution is to find individuals from which better systems can be constructed. Neef et al (1999) adapted the niche radius with competitive coevolution in multiobjective optimization. Lohn et al (2002) embodied competitive coevolution in the multiobjective optimization, which had two populations, the population of candidate solutions and the target population consisting of target objective vectors. Liu et al (2001) used cooperative coevolution to speed up convergence rates of fast evolutionary programming on large-scale problems whose dimension ranged from 100 to 1000. Keerativuttiumrong et al (2002) tried to extend the cooperative coevolution to multiobjective optimization by evolving each species with the multiobjective genetic algorithm (Fonseca and Fleming, 1993). This combination of coevolution and MOGA is rather elementary and its performance is not satisfactory.

Although evolutionary algorithm is a powerful tool, it needs to perform a large number of function evaluations in the evolution process. The computational cost involved in terms of time and hardware often become surprisingly burdensome as the size and complexity of the problem increases. One promising approach to overcome the limitation is to exploit the inherent parallel nature of EA by formulating the problem into a distributed computing structure suitable for parallel processing, i.e., to divide a task into subtasks and to solve the subtasks simultaneously using multiple processors. This divide-and-conquer approach has been applied to EA in different ways and many parallel EA implementations have been reported in literatures (Cantú-Paz, 1998; Goldberg, 1989b; Rivera, 2001). As categorized by Rivera (2001), there are four possible strategies to parallelize EAs, i.e., global parallelization, fine-grained parallelization, coarse-grained parallelization, and hybrid parallelization. Because the communication amount in coarse-grained parallelization is small compared with other parallelization strategies, it is a suitable computing model for distributed computer network where the communication speed is limited.

The major concern of this paper is to introduce the idea of cooperative coevolution into multiobjective optimization and exploit the intrinsic parallelism of coevolution. A distributed cooperative coevolutionary algorithm (DCCEA) is proposed and implemented based on this idea to search the Pareto front effectively and reduce the runtime in a Java-based distributed system framework named Paladin-DEC (Tan *et al*, 2002a, 2002b). The proposed approach is different from the existing cooperative coevolutionary algorithms in its adaptations to multiobjective optimization, such as the archive, extending operator, and the distributed computing structure.

The remainder of this paper is organized as follows. Section 2 presents the design of DCCEA for multi-

objective optimization. Section 3 provides the extensive case studies. Finally, conclusions are drawn in section 4.

2 The Design of Distributed Cooperative Coevolutionary Algorithm for Multi-objective Optimization

2.1 Extending Cooperative Coevolution into Multi-objective Optimization

Given a single objective optimization problem with n parameters, each parameter is assigned a subpopulation, and these n subpopulations coevolve the individuals in each of them (Potter and De Jong, 1994; Liu et al, 2001). Our algorithm adopts the idea of assigning one subpopulation to each parameter and adapts this idea to multiobjective optimization problems where multiple non-dominated solutions are aimed at. Figure 1 depicts the principle of cooperation and rank assignment in DCCEA. Here, individuals in subpopulation i cooperate with representatives of other subpopulations to form complete solutions.

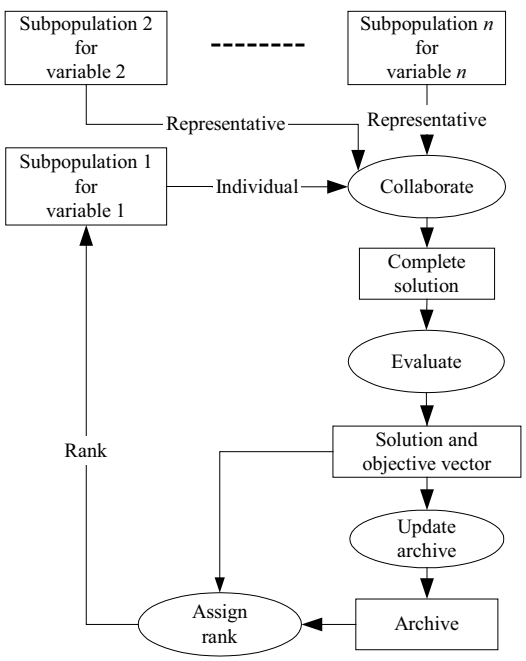


Figure 1. Cooperation and rank assignment in DCCEA

Each subpopulation only optimizes one parameter and an individual in a subpopulation is just a component of a complete solution. Here, we let the best individual in a subpopulation be the representative of the subpopulation. To evaluate an individual in a subpopulation, we first combine this individual with representatives of other subpopulations to form a complete solution. Then this complete solution is mapped into an objective vector by the objective functions. This objective vector can be used to evaluate how well the selected individual cooperates with other subpopulations to produce good solutions.

Multiobjective optimization requires finding a solution set whose objective vectors approximate the true Pareto front. How to obtain multiple solutions in one run using cooperative coevolution? Our approach is to introduce an archive into the algorithm, which stores and updates the non-dominated solutions found so far. This archive at the final generation will be output as the optimal solution set. The archive can also be regarded as an elitism mechanism since it preserves the best solutions so far. Moreover, the archive is used as a comparison set in the rank assignment of individuals in subpopulations after these individuals obtain their objective vectors through collaboration. A canonical Pareto ranking scheme is applied in the rank assignment and the rank of an individual i can be given by

$$rank(i) = 1 + n_i \tag{1}$$

where n_i is the number of archive members dominating the individual i in the objective domain.

The archive has a maximum size, which can be set according to the required number of solutions. Each time a complete solution is evaluated by the objective functions, it will update the archive according to its objective vector. When the solution is non-dominated with respect to the archive, it is added into the archive and the archive members dominated by it are removed. Once the archive is full, a truncation method based on niching will be activated to replace the most crowded archive member with the new non-dominated solution and keep the maximum size and diversity of the archive.

To distribute the solutions evenly along the Pareto front, the technique of niche induction by means of a sharing function is often implemented in MOEAs. Let $d(i, j)$ be the Euclidean distance between objective vectors of individuals i and j. The neighborhood size is defined in terms of Euclidean distance and specified by the so-called niche radius ω_{share}. The sharing function is defined as follows:

$$sh(d) = \begin{cases} (1 - d/\omega_{share})^2 & \text{if } d < \omega_{share} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

And the niche count function is defined according to the sharing function:

$$nc(i) = \sum_j sh(d(i, j)) \tag{3}$$

The niche radius is difficult to determine in practice because *a-priori* knowledge about the shape of the Pareto front for most problems is not available. Tan et al. (1999) proposed a mechanism of dynamic sharing computation that can adaptively update the niche radius along with the evolution. The sharing distance $\omega_{share}^{(n)}$ at generation n in term of the objective number m, the population size N, and the diameter of the population $d^{(n)}$ is given by

$$\omega_{share}^{(n)} = \lceil N^{1/(1+m)} \rceil d^{(n)} \tag{4}$$

The computation of diameter $d^{(n)}$ can be referred to (Tan et al., 2003). Formula (4) provides a good approach

to calculate niche radius, which does not require *a-priori* knowledge of the usually unknown trade-off surface. Besides the archive updating, the niche count is also applied in the tournament selection to generate the mating pool. In case two individuals have the same rank, the one with the less niche count wins in the tournament.

Furthermore, a new feature is introduced to improve the smoothness and spread of the solution set. Ordinarily, the underpopulated regions are the gaps or boundaries of the archive, which should be given more attention if the archive is expected to cover the true Pareto front by as much as possible. To make these unobvious regions outstanding, the extending operator could guide the evolutionary search into these areas. The archiving scheme plays a critical role in the realization of the extending operation. Firstly, since complete solutions are all stored in the archive, the subpopulations have no pressure to keep the diversity of their own individuals so that they can adaptively focus their search in the regions that are not explored thoroughly. Secondly, by extracting the information of the solution distribution from the archive, archive members in the most underpopulated regions will be cloned and added into the subpopulations. Thus, these members might have a great chance to be selected into the mating pool. Detailed description of the extending operator is given in figure 2.

The Extending Operator:
Let n be the number of clones
Step 1) If the archive is not full, exit.
Step 2) Calculate the niche count of each member in the archive. Then find the member with the smallest niche count. This member resides in the most underpopulated region.
Step 3) Clone n copies of this archive member to the subpopulations. Here, each part of this member is cloned into the corresponding subpopulation.

Figure 2. The procedure of the extending operator

2.2 The framework of DCCEA

DCCEA adopts the coarse-grained parallelization strategy of EAs. To fit into a distributed scenario, the design of DCCEA should concern several features of distributed computing such as variant communication overhead, different computation speed and network restrictions. The toy model with six subpopulations and three peers is given in figure 3 to illustrate the design idea of DCCEA.

As shown in figure 3, each parameter of the problem is assigned a subpopulation. In a distributed scenario, these subpopulations are further partitioned into a number of groups, which is determined by the available number of peers. In figure 3, the six subpopulations are divided into 3 groups and each of them is assigned to a peer computer. Each peer has its own archive and representative and evolves its subpopulations sequentially. Inside a peer, the complete solution generated through collaboration continuously updates the peer archive. The subpopulations in the peer update the corresponding peer representatives once every cycle. The cooperation among peers is indirectly achieved through the exchanges of archive and representatives between peers and a central server. In distributed scenario, the communication time among peers is a conspicuous part of the whole run time. To reduce the communication overhead, the exchange of archive and representatives between one peer and the central server occurs once every several generations. The number of generations between two exchanges is called the exchange interval.

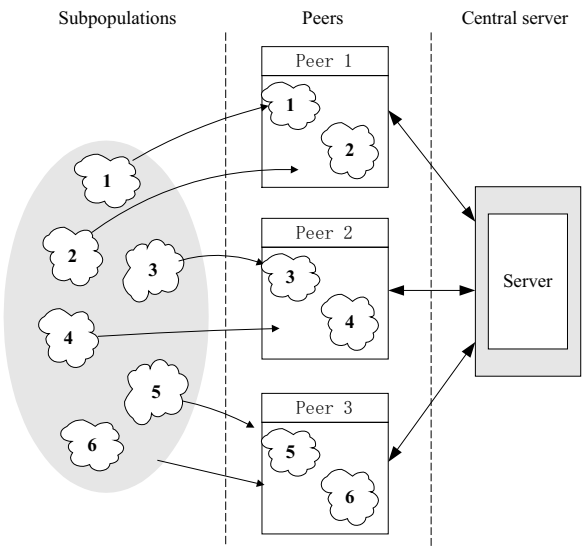


Figure 3. The model for DCCEA

Generally the peers are not identical and the cooperation among peers becomes ineffective if there are big differences in the evolution progresses of nodes. In such case, the bad cooperation among nodes deteriorates the performance of DCCEA. To keep the peers cooperate well in the evolution, these peers should be synchronized every some generations. Here, the synchronization interval is defined as the number of generations between two synchronizations. The exchange interval and synchronization interval can be fixed or adaptively determined along the evolution.

2.3 The Implementation of DCCEA

The implementation of DCCEA is imbedded into the distributed computing framework named Paladin-DEC (Tan et al, 2002a, 2002b) , which is built upon the foundation of Java technology offered by Sun Microsystems and is equipped with application programming interfaces (APIs) and technologies from J2EE. As shown in figure 4, the Paladin-DEC software consists of two main blocks, i.e., the servant block and workshop block that are connected by RMI-IIOP (Remote Method Invocation over Internet Inter-ORB Protocol). The servant functions as an information center and backup station through which peers can check their identifications or restore their working status. The workshop is a place where peers (free or occupied) work together in groups, e.g., the working peers are grouped together to perform

the specified task, while the free ones wait for the new jobs to be assigned.

The servant contains three different servers, i.e., logon server, dispatcher server, and database server. The logon server assigns identification to any registered peers. It also removes the information and identification of a peer when it is logged off as well as synchronizes the peer's information to the dispatcher server. The dispatcher server is responsible for choosing the tasks to be executed, the group of peers to perform the execution, and to transfer the peers' information to/from the database server. The dispatcher server also synchronizes the information, updates the peer's list, and informs the database server for any modification. Whenever there is a task available, the dispatcher server will transfer the task to a group of selected peers.

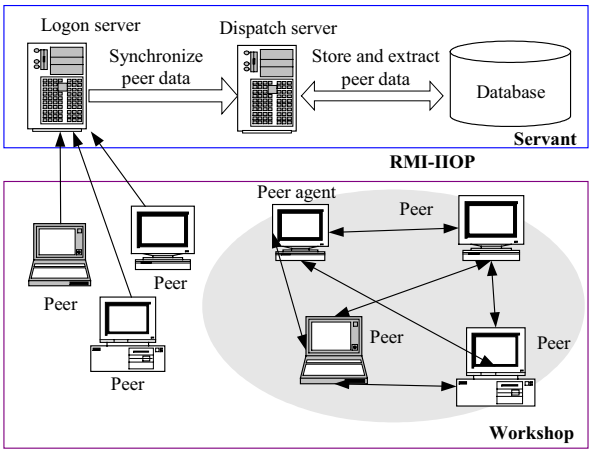


Figure 4. Schematic framework of Paladin-DEC software

The working process of a peer begins when the peer (or client) is started and logons to the server, which is realized by sending a valid email address to the server. The peer computer will then be pooled and waiting for the task to be assigned by the server. Once a peer detects that a task is assigned, it will extract the information from the server such as class name and path as well as the http server address and then load the class remotely from the server. If the class loaded is consistent to the Paladin-DEC system, it will be allowed to initiate the computation procedure. When the peer is allowed to start the computation procedure, it first initialize the parameters, such as generation number, subpopulation groups, subpopulation size, crossover rate, mutation rate. Then the peer creates the subpopulations assigned to it. Synchronization is crucial to DCCEA to achieve good cooperation among peers. When a peer reaches a synchronization point, it suspends its evolution until the server signals that all the peers have reached the synchronization point. In each generation, the peer will check whether it is time to exchange the archive and representatives between the peer and the server. If the conditions of exchange are satisfied, the peer will initiate a session in the server that retrieves the archive and representatives of the peer, then updates the server archive with the peer archive and updates the server representatives that correspond to the peer. In the side of peer, the peer will obtain the new server archive and server representatives and replace its archive and representatives. After these steps, the peer evolves its subpopulations sequentially one generation. If the peer meets the termination conditions, it will initiate a session to submit the results, and afterwards, restore itself to the ready status. If the user cancels the running job, those peers involved in the job will stop the computation and set themselves to the ready status.

2.4 Workload Balancing

Since the processing power and specification for various computers in a network might be different, the feature of work balancing that ensures the peers are processed in a similar pace is needed in the DCCEA is important because the total computation time is decided by the peer that finished the work last, and if the peer with the least computational capacity is assigned with the most heavy workload, not only would longer time be required but the bad cooperation among nodes deteriorates the performance of DCCEA. Intuitively, work balancing for a distributed system could be difficult due to the fact that the working environment in a network is often complex and uncertain. The DCCEA resorts to a simple work balancing strategy by assigning the workload to the peers according to their respective computational capabilities. As stated in section 2.3, when a peer is first launched, it uploads its configuration information, which could be accessed by the servant. The hardware configuration of the peer is recorded in the information file, such as the CPU speed, RAM size, etc. After reading the information file, the dispatch server performs a simple task scheduling and assigns different tasks to the respective peers according to their computational capabilities.

3 Case study

In this section, we first describe the test problems used in the comparisons. Next, two performance metrics for multiobjective optimization are described and defined. Then, extensive simulations of the algorithms are performed on these test problems.

3.1 The Test Problems

Five test problems are resorted to validate the performance of DCCEA. Table 1 summarizes features of these test problems and these problems are defined in table 2. They include important characteristics and are suitable to validate the effectiveness of MOEAs. (Knowles and Corne, 2000; Corne et al, 2000; Deb, 2002; Zitzler et al, 1999, 2000, 2001), have used these problems in the validations of their algorithms. Therefore these problems should be a good test suite for a fair comparison of different multiobjective algorithms.

These problems were designed using Deb's scheme by Zitzler et al (2000), and were used in a performance comparison of eight well-known MOEAs. Each of these test problems is structured in the same manner and consists itself of three functions (Deb, 1999):

$$\begin{aligned} &\textit{Minimize } T(x) = (f_1(x_1), f_2(x)) \\ &\textit{subject to } f_2(x) = g(x_2,\cdots,x_m)h(f_1(x_1), g(x_2,\cdots,x_m)) \\ &\textit{where } x = (x_1,\cdots,x_m) \end{aligned} \tag{5}$$

The definitions of f_1, g, h in ZDT1, ZDT2, ZDT3, ZDT4 and ZDT6 are listed in table 2.

Table 1. Features of the test problems

Test Problem	Features
ZDT1	Pareto front is convex.
ZDT2	Pareto front is non-convex.
ZDT3	Pareto front consists of several noncontiguous convex parts.
ZDT4	Pareto front is highly multi-modal and there are 219 local Pareto fronts.
ZDT6	The Pareto-optimal solutions are non-uniformly distributed along the global Pareto front. The density of the solutions is lowest near the Pareto-optimal front and highest away from the front.

Table 2. Definitions of f_1, g, h in ZDT1, ZDT2, ZDT3, ZDT4 and ZDT6

ZDT1

$$\begin{aligned} &f_1(x) = x_1 \\ &g(x_2,\cdots,x_m) = 1 + 9\sum_{i=2}^{m} x_i/(m-1) \\ &h(f_1,g) = 1 - \sqrt{f_1/g} \\ &\textit{subject to } x = (x_1,\cdots,x_m), m = 30, \textit{ and } x_i \in [0,1]. \end{aligned} \tag{6}$$

ZDT2

$$\begin{aligned} &f_1(x) = x_1 \\ &g(x_2,\cdots,x_m) = 1 + 9\sum_{i=2}^{m} x_i/(m-1) \\ &h(f_1,g) = 1 - (f_1/g)^2 \\ &\textit{subject to } x = (x_1,\cdots,x_m), m = 30, \textit{ and } x_i \in [0,1]. \end{aligned} \tag{7}$$

ZDT3

$$\begin{aligned} &f_1(x) = x_1 \\ &g(x_2,\cdots,x_m) = 1 + 9\sum_{i=2}^{m} x_i/(m-1) \\ &h(f_1,g) = 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1) \\ &\textit{subject to } x = (x_1,\cdots,x_m), m = 30, \textit{ and } x_i \in [0,1]. \end{aligned} \tag{8}$$

ZDT4

$$\begin{aligned} &f_1(x) = x_1 \\ &g(x_2,\cdots,x_m) = 1 + 10(m-1) + \sum_{i=2}^{m}(x_i^2 - 10\cos(4\pi x_i)) \\ &h(f_1,g) = 1 - \sqrt{f_1/g} \\ &\textit{subject to } x = (x_1,\cdots,x_m), m = 10, x_1 \in [0,1], \\ &\textit{and } x_2,\cdots,x_m \in [-5,5]. \end{aligned} \tag{9}$$

ZDT6

$$\begin{aligned} &f_1(x) = 1 - \exp(-4x_1)\sin^6(6\pi x_1) \\ &g(x_2,\cdots,x_m) = 1 + 9((\sum_{i=2}^{m} x_i)/(m-1))^{0.25} \\ &h(f_1,g) = 1 - (f_1/g)^2 \\ &\textit{subject to } x = (x_1,\cdots,x_m), m = 10, x_i \in [0,1]. \end{aligned} \tag{10}$$

3.2 Metrics of Performance

Two different quantitative performance measures for multiobjective optimization are used in this study. They are referred from (Deb, 2002) and modified slightly by us.

The metric of generational distance is a value representing how "far" the PF_{known} is from PF_{true} and is defined as:

$$GD = (\frac{1}{n}\sum_{i=1}^{n} d_i^2)^{1/2} \tag{11}$$

where n is the number of members in PF_{known}, d_i is the Euclidean distance (in objective space) between member i in PF_{known} and its nearest member in PF_{true}. The smaller the generational distance is, the closer the PF_{known} is to PF_{true}.

The metric of spacing measures how "evenly" members in PF_{known} distribute. It is defined as:

$$S = [\frac{1}{n}\sum_{i=1}^{n}(d_i - \bar{d})^2]^{1/2} / \bar{d} \tag{12}$$

where $\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i$, n is the number of members in PF_{known}, d_i is the Euclidean distance in objective space between the member i in PF_{known} and its nearest member in PF_{known}. The smaller the spacing is, the more evenly members in PF_{known} distribute.

3.3 Simulation Results of DCCEA

DCCEA is integrated into the framework of Paladin-DEC. The test environment for DCCEA consists of 11 PCs in the campus LAN. Table 3 gives configurations of the 11 PCs. The server part of Paladin-DEC runs on the PIV 1600/512. Peers of Paladin-DEC run on other PCs.

Table 3. Running Environment of DCCEA

PC	Configuration CPU (MHz)/RAM (MB)
1	PIV 1600/512
2	PIII 800/ 512
3	PIII 800/ 512
4	PIII 800/ 256
5	PIII 933/384
6	PIII 933/128
7	PIV 1300/ 128
8	PIV 1300/ 128
9	PIII 933/ 512
10	PIII 933/ 512

11	PIII 933/256

DCCEA is proposed to exploit the parallelism among subpopulations. Because test problems, ZDT1, ZDT2, ZDT3, ZDT4 and ZDT6, have a large number of decision variables, they are suitable to test DCCEA on the capacity of effectively accelerating the multiobjective optimization. The parameter configurations of DCCEA can be seen in table 4.

Table 4. The Configurations of DCCEA

Populations	Subpopulation size 20; archive size 100
Chromosome length	30 bits for each variable.
Selection	Binary tournament selection
Crossover operator	Uniform crossover
Crossover rate	0.8
Mutation operator	Bit-flip mutation
Mutation rate	$2/L$, where L is the chromosome length.
Number of evaluations	120,000
Exchange interval	5 generations
Synchronization interval	10 generations

Thirty independent runs are performed with random initial population to minimize any bias in the simulations. The median runtime of 30 runs is listed in table 5 and visualized in figure 5. It can be seen that the median runtime goes down as the number of peers increased. In the case of ZDT1, the median runtime for 6 peers (each peer with 6 subpopulations) is 96 seconds, which is about 1/3 of the 270 seconds used by 1 peer (each peer with 30 subpopulations). It is also evident that 6 peers are enough for the acceleration of runtime. When there are more than 6 peers, the increment of communication cost counteracts the reduction of computation cost of each peer and saturation of acceleration is nearly achieved.

Table 5. Median runtime of 30 runs with respect to the numbers of peers (unit: second)

Number of peers	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
1	270	242	189.5	209	138
2	177.5	142.5	128.5	170	137
3	134	121.5	101	142	124
4	120	109.5	97	139	121
5	109	90	88	134	121
6	96	80	67	123	108
7	94	73	68.5	111	110
8	80	74	65	115	109.5
9	78	72	64	114	109.5
10	78	76	68	115	110.5

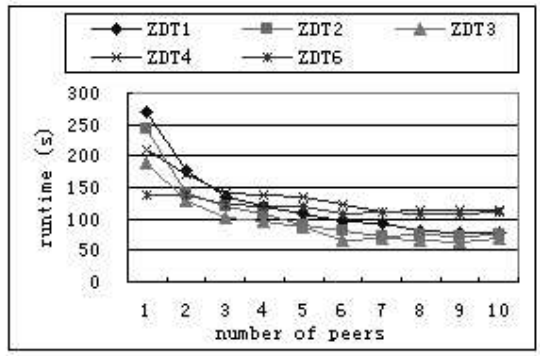


Figure 5. Median runtime of 30 runs with respect to the numbers of peers

The median metrics of 30 runs are summarized in figure 6 and 7. It can be seen that the median metrics have no distinct change in spite of some small fluctuations on the curve for the five test problems as the number of peers increases. So it can be said that the DCCEA can effectively reduce the runtime while not deteriorating the performances as the number of peers increases.

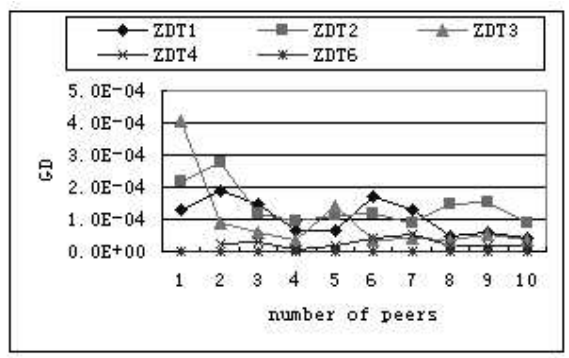


Figure 6. Median generational distance of 30 runs with respect to the numbers of peers

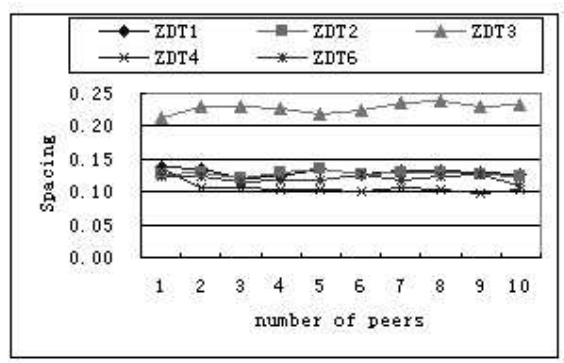


Figure 7. Median spacing of 30 runs with respect to the numbers of peers

3.4 Comparisons with Other Works

In this section, simulations are carried out to validate the performance of DCCEA through comparisons with other

recently presented evolutionary multiobjective optimization methods. In the comparision, DCCEA with 6 peers acts as the basis of our algorithm. The evolutionary multiobjective optimization methods include PAES (Knowles and Corne, 2000), PESA (Corne et al. 2000), NSGAII (Deb et al., 2002), SPEA2 (Zitzler and Thiele 2001) and IMOEA (Tan et al., 2001), which are all recently presented algorithms in literatures. Although such comparisons are not meant to be exhaustive, it provides a good basis to assess the DCCEA.

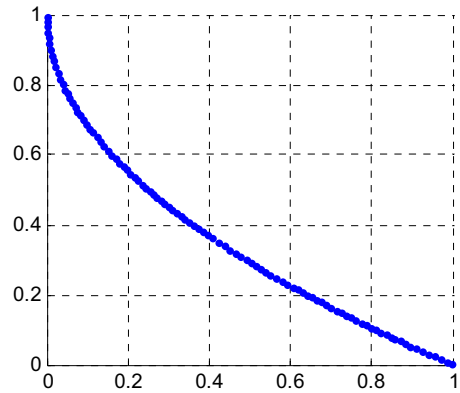
In order to guarantee a fair comparison, all MOEAs considered are implemented with the same binary coding, binary tournament selection, uniform crossover, and bit-flip mutation. Also, the number of evaluations in each run is fixed as 12,000. Table 6 lists other configurations of these algorithms that are very common in studies of MOEAs. In the experiment, each algorithm runs 30 times on each test function to study the statistical performance.

Table 6. Configuration of PAES, PESA, NSGAII, SPEA2, IMOEA.

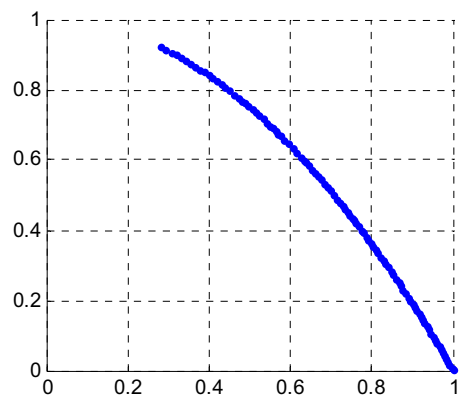
Population	Population size 1 in PAES; population size 100 in PESA, NSGAII, SPEA2; initial population size 20, maximum population size 100 in IMOEA. Secondary population (or archive) size is 100 for all the algorithms.
Chromosome	Binary coding, 30 bits for each variable.
Crossover operator	Uniform crossover
Crossover rate	0.8
Mutation operator	Bit-flip mutation
Mutation rate	$2/L$, where L is the chromosome length.
Number of evaluations	120,000
Hyper-grid size	32 x 32 grid in PAES and PESA.

Figure 8 gives the generated Pareto front for ZDT4 and ZDT6 by DCCEA with 6 peers. The results approximate the true Pareto front very well and our eyes almost cannot tell the difference between the generated and the true Pareto fronts.

Table 7 and 8 summarizes the simulation results with respect to the two performance metrics. For almost all the test problems and metrics, the performance of PAES is the worst. The reason may be that PAES is just a non-population based local search algorithm where mutation acts as a local search method. With respect to generational distance, DCCEA is outstanding in ZDT4 and ZDT6. Because of the many local Pareto fronts of ZDT4 and non-uniform distribution of ZDT6, these two problems are difficult for MOEAs. It means that DCCEA has a strong ability to escape from harmful local optima. For other problems, the performance of DCCEA on generational distance is also very competitive. Concerning the metrics of spacing, DCCEA has shown to be the best in all cases. This means that DCCEA has an excellent ability to maintain diversity of the solution set. In brief, the DCCEA is strongly competitive in comparison with other MOEAs.



(a) ZDT4



(b) ZDT6

Figure 8. Generated Pareto front for ZDT4 and ZDT6 by DCCEA with 6 peers.

Table 7. Median generational distance of 30 runs

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
DCCEA	1.70E-04	1.15E-04	1.05E-04	4.35E-05	3.42E-08
PAES	4.93E-03	4.95E-03	1.94E-02	1.00E-00	1.41E-01
PESA	3.93E-06	3.89E-06	1.19E-04	7.70E-01	2.71E-02
NSGAII	1.49E-03	7.44E-04	2.27E-03	7.66E-01	7.50E-03
SPEA2	1.95E-03	2.23E-03	2.93E-03	7.79E-01	1.08E-02
IMOEA	2.25E-04	1.43E-04	1.26E-03	7.80E-01	5.07E-07

Table 8. Median spacing of 30 runs

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6
DCCEA	0.1272	0.1273	0.224	0.0994	0.1251
PAES	0.901	0.9658	2.0254	2.4148	3.4356
PESA	0.5148	0.5	0.6542	0.6119	1.6997
NSGAII	0.4339	0.3081	0.4844	0.481	0.6184
SPEA2	0.5717	0.589	0.6582	0.6347	0.8389
IMOEA	0.3449	0.2215	0.2502	0.7609	0.5491

4. Conclusion

This paper has applied the coevolution mechanism into multiobjective optimization. A number of subpopulations represent different decision variables and evolve independently. The cooperation among subpopulations is achieved through the sharing of archive and representatives of subpopulations. Such a loosely coupled paradigm can be easily formulated into a distributed computing structure suitable for parallel processing. Computational results show that the DCCEA can dramatically reduce the runtime while keeping the performance as the peer number increases.

Bibliography

E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles*, *Reseaux et Systems Repartis*, Paris: Hermes, vol. 10, no. 2, pp. 141-171, 1998.

D.W. Corne, J.D. Knowles, and M.J. Oates, "The Pareto envelope-based selection algorithm for multiobjective optimization," in *Proceedings of the Parallel Problem Solving from Nature VI Conference (PPSN VI)*, pp. 839-848, Springer, 2000.

K. Deb, "Multiobjective genetic algorithms: problem difficulties and construction of test problem," *Evolutionary Computation*, vol. 7, no. 3, pp. 205-230, 1999.

K. Deb, *Multiobjective optimization using evolutionary algorithms*. John Wiley & Sons, New York, 2001.

K. Deb, et al, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp.182-197, 2002.

C.M. Fonseca, and P.J. Fleming, "Genetic algorithms for multiobjective optimization, formulation, discussion and generalization," in *Proceeding of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 416-423, 1993.

D.E. Goldberg, and J. Richardson, "Genetic algorithms with sharing for multi-modal function optimization," in *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 41-49, 1987.

D.E. Goldberg, "Sizing populations for serial and parallel genetic algorithms", in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 70-79, 1989.

N. Keerativuttiumrong, N. Chaiyaratana and V. Varavithya, "Multiobjective co-operative co-evolutionary genetic algorithm," in *Proceedings of the Parallel Problem Solving from Nature VII Conference (PPSN VII)*, pp. 288-297, 2002.

J.D. Knowles, and D.W. Corne, "Approximating the non-dominated front using the Pareto archived evolution strategy," *Evolutionary Computation*, vol. 8, no. 2, pp. 149-172, 2000.

Y. Liu, X. Yao, et al, "Scaling up fast evolutionary programming with cooperative coevolution," in *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 2, pp.1101-1108, 2001.

J.D. Lohn, W.F. Kraus and G.L. Haith, "Comparing a coevolutionary genetic algorithm for multiobjective optimization," in *Proceedings of the 2002 Congress on Evolutionary Computation*, v. 2, pp. 1157-1162, 2002.

M. Neef, D. Thierens, and H. Arciszewski, "A case study of a multiobjective recombinative genetic algorithm with coevolutionary sharing," in *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 796-803, 1999.

M.A. Potter, and K.A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proceedings of the Parallel Problem Solving from Nature III Conference (PPSN III)*, pp.249–257, Berlin, Germany, 1994.

M.A. Potter, and K.A. De Jong, "Cooperative coevolution: an architecture for evolving coadapted subcomponents," *Evolutionary Computation*, vol. 8, no. 1, pp.1-29, 2000.

W. Rivera, "Scalable parallel genetic algorithms", *Artificial Intelligence Review*, vol. 16, pp.153-168, 2001.

C.D. Rosin, and R.K. Belew, "New methods for competitive coevolution," *Evolutionary Computation*, vol. 5, no. 1, pp. 1-29, 1997.

J.D. Schaffer, "Multiple-objective optimization using genetic algorithms," in *Proceedings of the First International Conference on Genetic Algorithms*, pp. 93-100, 1985.

K.C. Tan, T.H. Lee, and E.F. Khor, "Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 6, pp. 565-588, 2001.

K.C. Tan, T.H. Lee, J. Cai, and Y.H. Chew, "Automating the drug scheduling of cancer chemotherapy via evolutionary computation," in *Proceedings of the 2002 Congress on Evolutionary Computation*, pp. 908-913, 2002a.

K.C. Tan, A. Tay, and J. Cai, "Design and implementation of a distributed evolutionary computing software," *IEEE Transactions on System, Man and Cybernetics: Part C*, 2002b.

K.C. Tan, E.F. Khor, T.H. Lee, and R. Sathikannan, "An evolutionary algorithm with advanced goal and priority specification for multiobjective optimization," *Journal of Artificial Intelligence Research*, vol. 18, pp.183-215, 2003.

E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: empirical results," *Evolutionary Computation*, vol. 8, no. 2, pp.173-195, 2000.

E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: improving the strength Pareto evolutionary algorithm," *Technical Report 103*, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Switzerland, May 2001.