2011 Third International Conference on Knowledge and Systems Engineering

A Parallel Cooperative Coevolution Evolutionary Algorithm

Chi Cuong VU
Department of Networking
Vinh University
Vinh, Nghe An, Vietnam
cuongvcc@gmail.com

Huu Hung NGUYEN
Department of Software Engineering
Le Quy Don Technical University
100 Hoang Quoc Viet St, Hanoi, Vietnam
hunghvktqs2003@gmail.com

Lam Thu BUI
Department of Software Engineering
Le Quy Don Technical University
100 Hoang Quoc Viet St, Hanoi, Vietnam
lam.bui07@gmail.com

Abstract

Evolutionary algorithms (EAs) have been widely applied to solve many numerical and combinatorial optimization problems. A special paradigm of EAs has been promoted allowing several populations to co-evolve together. During evolution process these populations can be either cooperative or competitive. The cooperative co-evolution evolutionary algorithm (CCEA) has shown a great deal in solving large and complex problems. However, there are limited studies on parallelizing cooperative co-evolution evolutionary algorithms. In this paper, we propose an approach combining CCEA with a synchronous parallel model. The design especially facilitates solving large scale problems. We conducted a preliminary investigation with several experiments on benchmark large scale problems. The experimental results indicated a promising performance of the proposed algorithm on the selected problems.

1. Introduction

The evolutionary algorithms (i.e. the genetic algorithm, originally proposed by Holland [4]) have been widely applied to a variety of problem domains including multimodal function optimization, machine learning and evolution of complex structures. However, there still exist several situations where EAs do not show advantages; for example, when solving large scale problems EAs suffer the effect of "the curse of dimensionality". CCEA has been proposed by [7] for tackling this issue where problems are decomposed into several parts and each is solved by an individual EA.

When the complexity of problems increases (ie.e the dimensionality), it causes longer processing time. For this, parallel algorithms may be useful in reducing the computational overheads. With problems having large dimensionality, decomposition is suitable scheme for a parallel process in which each CPU executes a designated part of work. In fact, it has been demonstrated as a good methodology with an island model [14]. In the light of this research direction, we propose a master/slave model for parallelizing CCEA where master is in charge of decomposing problems and maintaining the best solution. To our best of knowledge, it is the first attempt to use master/slave architecture for CCEA. The experimental results obtained on testing three benchmark problems [11] showed a promising performance of this master/slave model.

The remainder of the paper is organized as follows. In section 2, we provide a summary of cooperative coevolution and different types of parallel models. In section 3, a description of the proposed master-slave model for CCEA is given. Section 4 is dedicated for experimental results and analysis when we applied our parallel cooperative co-evolution on three benchmark functions.

2. Cooperative Co-evolution and Parallel Computing

Conventional cooperative co-evolution was proposed by Potter [7, 6, 16] as a framework to tackle complex problems with high dimensionality. It simulates the evolution process of an ecosystem consisting of two or more species. In nature, the species are genetically isolated - meaning that individuals only mate with other members of their species. Mat-

978-0-7695-4567-7/11 $26.00 © 2011 IEEE
DOI 10.1109/KSE.2011.16

ing restrictions are enforced simply by evolving the species in separate populations. The species interacts with one another within a shared domain model and has a cooperative relationship. In the case of simulating optimization processes, a number of populations (each represents a species) are generated and evolve concurrently. Each population evolves on a part of the problem (a subcomponent). During evolution time, they can share information among each others. A complete solution for the problem consists of all subcomponents, which are representatives from all species. That is the way they co-operate during evolution. The representative of a population can be either the current best individual of the population or a randomly-selected one. Readers ar referred to [15] for a detailed description. Cooperative co-evolution experiences three main steps:

1. **Decomposing** the problem: The vector of problem parameters is decomposed into several subcomponents. They are handled by certain evolutionary algorithms.

2. **Optimizing** subcomponents: Use a certain EA for evolving each subcomponent separately until the stopping criteria are satisfied. This means a population will be used for optimizing a subcomponent.

3. **Co-adapting** subcomponents: During the evolution process, populations exchange information among each other and co-adapt towards the optimal solution.

The first step (decomposition) is quite essential. This "divide and conquer" strategy not only affects algorithm performance, but also the convergence speed and quality of the final solution. The problem is divided into into several subcomponents. For each subcomponent, a population is generated and evolves. This means they are evolving on different problems. The complete solution is built from all subcomponents.

Assuming that there are *s* populations (called P_i), the cooperative co-evolution using to solve high-dimensional problems is summarized as follows:

cycle = 0

For i:=0 to s

1) Initialize( ) //Initialize randomly

2) Evaluate( )

While (termination criteria = false) do

1) cycle = cycle +1

2) For i:=0 to s

a. $P_i(cycle) = Select(P_i(cycle-1))$ //Select base on fitness

b. EAApply( )

c. Evaluate( )

There can be any kind of evolutionary algorithms for the procedure *EAApply()*. In this paper we use the Differential Evolution (DE), a special version of evolutionary algorithms dealing with real-value problems, and proposed

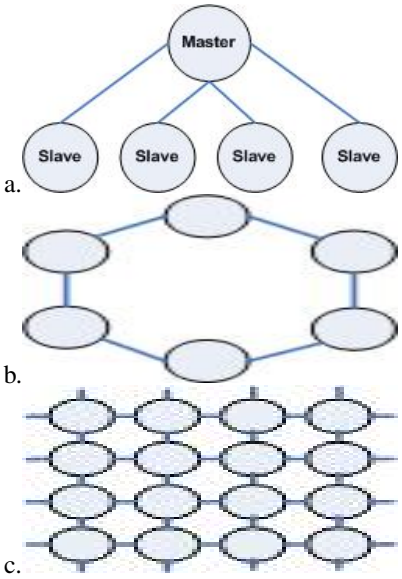

Figure 1. Models of parallel EAs: a-Master-Slave, b-Coarse grained, c-Fine grained

by Storn and Price [5, 9, 10, 8]. The classical DE will be shown as follow (assuming that a population consists of *PS* individuals and each individual is described by an n-dimensional vector *x*). For each x_i, we generated an offspring $x_i^{'}$ for the next generation.

1. Mutation: for each value $x_i^{'}$

$x_i^{'} = x_j + F(x_k - x_l)$ where *i,j,l,k* are integers and mutually different. *j,l,k* are randomly generated. The constant *F* is used to control the differential variation

2. Crossover: for each value $x_i^{''}$

Set $x_i^{''} = x_i^{'}$ if $U[0,1] \leq CR$ or $i = i_{rand}$ and if otherwise

Where $rand$ is a random number, $U[0,1]$ is a random number between 0 and 1, and CR is crossover rate.

3. Selection

$x^{'} = x^{''}$ if $f(x^{''}) \leq f(x^{'})$ else $x^{'} = x$.

The idea of parallelization is to divide computation into smaller components, and then compute them at different process units. According to Flynn's taxonomy [3], there are four parallel computer architectures, in which we consider a special one called multiple instructions, multiple data (MIMD). In MIMD, all processors can work at the same time. Processors may execute multiple instructions on different pieces of data. In this paper, we use MIMD to solve benchmark problems.

For parallel EA (PEA)[1], implementation can be classified into the following models:

The master slave model has a single population. Master node is designated to execute EAs (mutation, crossover and selection) while slaves calculate fitness of individuals (Fig 1a). This model is quite straightforward but requires a heavy traffic load since individuals are repeatedly sent over the interconnection channel. The coarse grained model is more sophisticated. They contain of a number of populations that can exchange individuals during evolution process and following a predefined interconnection pattern (see Fig 1b). The third type of PEA is a fine-grained model. The model has many nodes that form a two-dimensional rectangular grid (2D mesh) see Fig 1c, each individual is embedded to a grid point. Each point in 2D mesh represents a processor that evolves the embed individual. All members of population perform the evaluation of fitness simultaneously.

Several works related to parallel DE have been proposed in [13, 2]. They applied the island model to parallel DE. Note that they focused on parallel DE, but not co-evolution. To our best of knowledge, there have been two models being proposed in terms of parallelizing cooperative co-evolution. In [12], the authors introduced a hierarchical parallel paradigm for cooperative co-evolution in which each logic node is a group of computing nodes in stead of a single node. Its is designed specifically for multi-objective problems. In [14], an island model was used for cooperative co-evolution. Populations exchange information in a cyclic fashion.

3. Parallelizing Cooperative Co-evolution

We attempt to parallelize the cooperative co-evolution algorithm without changing the co-evolutionary feature. We propose to design a master/slave model for it. However, this model is quite different from the conventional master/slave model introduced above in the sense of the evolution role. It allows evolution of populations resided on slaves rather than just evaluating the fitness, while the master just acts a controller. In more details, the master processor is in charge of generating a number of populations, in which each population will evolve on a part of the problem or a subcomponent (hence having a lower dimension: assuming a balanced decomposition, if we have a problem with n dimensions and s slaves, then the dimension for each subcomponents will be n/s). Also it maintains a best individual overtime and we call it as the global best. After that, it sends to each slave:

- A population
- The global best individual.

In each slave, the population is evolved by using the DE algorithm for a finite number of times (we call it an evolution cycle). After an evolution cycle, each slave selects a local best individual as its representative and then sends back to the master for updating the global best. The procedure of updating is as follows:

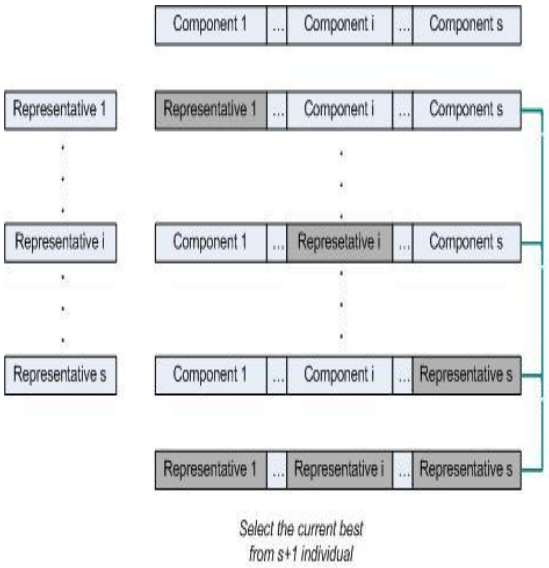

Figure 2. Selection the current best method at master

These representatives will replace their corresponding component in the current best solution respectively. If assuming that we have s slaves, we will have s new complete solutions. Also all representatives will be combined to form a solution. Hence we will have s+1 solutions and these solutions will be compared with the current best global best solution to find the new global best one. The more detailed description can be drawn from Figure 2. The master always performs checking the termination condition; if this criterion is met, the search process will finish. If not, master sends the best individual and demands all slaves to continue evolutional process.

Algorithm at master can be described as follows:

For i:=0 to s
1) Initialize( ) //Initialize randomly
2) SendToSlave( )
cycle = 0
GetBest(bestIndividual)
While termination criteria = false do
1) cycle = cycle +1
2) For j:=0 to s
a. SendToSlave(bestIndividual)
b. (Waiting for slaves)
c. ReceiveFromSlave(bestComponents)
3) Update(bestIndividual)

Algorithm at slaves i can be described as follows:

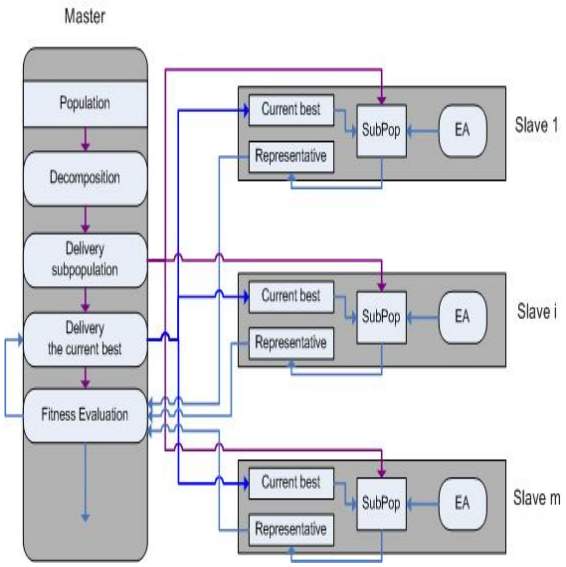

Figure 3. Parallel Cooperative Co-evolution Model

		No. Slaves	Time	Best
F1		1	9.416	8027.137
		2	4.518	149.299
		3	1.960	0.635
		4	2.661	0.141
		5	1.665	0.00283
F2		1	48.507	79476328.100
		2	28.462	2677892.884
		3	16.200	40788.236
		4	26.447	18195.565
		5	20.742	5804.950
F3		1	19.85	20.48
		2	9.159	12.65
		3	5.31	11.93
		4	7.11	5.70
		5	5.22	4.27

Table 2. Obtained results: each line shows the time to complete a run and the best individual found

ReceiveFromMaster( )
While terminatioin criteria = false
1) ReceiveFromMaster(bestIndividual)
2) GAApply( ) //with a predefined fitness evaluations
3) Evaluate( )
4) SendToMaster(bestComponent)
(Waiting for master)

As depicted in Figure 3, populations are delivered to slaves through an interconnection between processors. The populations are only sent from the master at *the first time*. Evolution of each species is executed separately and concurrently. The cooperation among species can be seen as contributing the best individuals for forming the complete solution of the problem, which is maintained on the master.

So, in terms of communication during the evolution process, only *s* best individuals are sent from the slaves to the master. Inversely, only one global best solution is sent from the master to slaves.

Note that, there is a tradeoff in controlling the number of slaves. Increasing the number of slaves will give better performance since the dimension of each sub-problem solving by a population will be reduced. However, if there are many slaves, it might cause a traffic problem on the interconnection channel such as the low bandwidth or bottleneck effect at the master processor. So we need to negotiate the number of slaves to achieve the balance. Master must wait for the best individuals of populations from slaves. The time that different slave using to complete an evolutional cycle is different. Therefore, master has to execute the parallel process *synchronously*.

4. Empirical Results and Discussion

To demonstrate the performance of our proposed model of parallelizing, we built an application using MPICH2 library which can support well for parallel computing and using C language. We have chosen three benchmark functions [11] and called them as F1, F2, F3 as test suits. The function detail will be showed in the following table:

We employed the classical DE (with CR=0.9 and F=0.3) for evolving populations. For all test suits, the population size was 100. We tested problem with dimensionality of 500 variables. We conducted experiments with different number of sub-populations: 1 (no decomposition), 2, 3, 4 and 5. Equivalently, the dimension in each case was reduced from 500 to 250, 166, 125, and 100. We used 200000 fitness evaluations. In addition, we also evaluated the optimal when the cycles increasing and population was subcomponent smaller. We used a clustering network (LAN 10/100Base-TX) with three PCs and each PC has two processors: Each pair of processors has the speed of 2.4, 2.0, and 2.2 Ghz respectively. So in total we had six processors (processing nodes). We always designated one processor for the master. Different computers have different process speeds. Each problem was tested for 20 times. All performance times are evaluated in second unit.

After conducting the experiments, we summarized the results in Table 2. Note that we used the case of one population (no decomposition) to compare with the parallelized ones. First of all, we can observe from the table that when

Prob.	Description	Min	Max	Opt. Solution
F1	$F_1(x) = \sum_{i=1}^{D} x_i$	100	-100	(0,0,...0)
F2	$F_2(x) = \sum_{i=1}^{D-1}(100(x_i^2 - x_{i+1})^2 + (x_i - 1))$	100	-100	(0,0,...0)
F3	$F_3(x) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i} - \exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i))) + 20$	100	-100	(0,0,...0)

Table 1. Description of testing problems

we increase the number of populations (or slaves), the total computing time was reduced accordingly. This is clearly understandable since the computation load was divided on more slaves.

There is an exception in the case of 3 slaves in which the computation time slightly shorter than that of the 4-slave case. This might be the result of overheads that came from threading management system of MPI. In our setup, 3-slave case used 2 PCs while 4-slave one used 3 PCs (since we need one more processor for the master). Therefore, it required more controlling work from the operating systems for 3 PCs than 2 PCs. Further, the time spending on a fitness evaluation is really short (about 0.01 millisecond), so in 4-slave case, it might consumed more time for management of the system took a significant portion of the total time.

Regarding the convergence results, the table obviously shows the advantage of using decomposition for solving the large scale problems. Our parallel model guaranteed this advantage. For F1, this is the easiest problem with a single and global optimal point. However, with a large dimension, DE's performance deteriorated badly with the best solution of 8027.137. With our parallel model, when the number of slaves increased, the quality of the best found solution also increased (0.00283). The similar finding is also applied for F2 and F3.

5. Conclusion

In this paper, we proposed a model of parallel cooperative co-evolutionary algorithm and applied to solve large scale optimization problems. In our model, a master will be in charge of decomposing the problem and maintaining the global best solution. Each slave will evolve on a sub-problem and they will exchange the best individuals to the master after each evolutionary cycle. We conducted experiments on three benchmark problems to validate the proposed model. The obtained results showed a clear advantage of using our parallel cooperative co-evolution system for solving large problems.

Acknowledgments

We acknowledge the financial support from Vietnam's National Foundation for Science and Technology (Development Grant 102.01-2010.12).

References

[1] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Trans. Evolutionary Computation*.

[2] J. Apolloni et al. Island based distributed differential evolution: An experimental study on hybrid testbeds. In *Eighth International Conference on Hybrid Intelligent Systems*, pages 696–701. IEEE, 2008.

[3] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, 21:948–960, September 1972.

[4] J. Holland. *Adaptation in Natural and Artificial Systems* . MIT Press, 1975.

[5] F. Neri and V. Tirronen. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review*, 33(1):61–106, 2010.

[6] M. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD thesis, George Mason University, 1997.

[7] M. Potter and K. De Jong. A cooperative coevolutionary approach to function optimization. *Parallel Problem Solving from NaturePPSN III*, pages 249–257, 1994.

[8] K. Price, R. Storn, and J. Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Verlag, 2005.

[9] R. Storn. System design by constraint adaptation and differential evolution. *Evolutionary Computation, IEEE Transactions on*, 3(1):22–34, 1999.

[10] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[11] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore, 2005.

[12] K. Tan, Y. Yang, and C. Goh. A distributed cooperative coevolutionary algorithm for multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 10(5):527–549, 2006.

[13] D. Tasoulis, N. Pavlidis, V. Plagianakos, and M. Vrahatis. Parallel differential evolution. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 2023–2029. IEEE, 2004.

[14] M. Weber, F. Neri, and V. Tirronen. Shuffle or update parallel differential evolution for large-scale optimization. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, pages 1–19.

[15] R. Wiegand. *An analysis of cooperative coevolutionary algorithms*. PhD thesis, Department of Computer Science, George Mason University, 2003.

[16] Z. Yang, K. Tang, and X. Yao. Large scale evolutionary optimization using cooperative coevolution. *Inf. Sci.*, 178:2985–2999, 2008.