Parallel Implementations of Coevolutionary Multi-objective Evolutionary Algorithms

Mark P. Kleeman* and Gary B. Lamont [*]
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Wright Patterson Air Force Base, Dayton, OH 45433

Key Words: Coevolution, MOEA, Parallel algorithm

Abstract

Multi-objective Evolutionary Algorithms (MOEAs) are a popular tool for researchers. This is because many real world problems have multiple objectives that need to be optimized and MOEAs have been shown to perform well in a variety of search landscapes. One area of MOEA research that is still in its infancy is investigating the application coevolutionary techniques to the algorithm. Since 1999, only a handful of researchers have explored the idea of combining coevolution with MOEAs. This paper explores what many of those researchers has done in the field of Coevolutionary MOEAs (CMOEA) and looks at the parallel techniques that can be used to enhance the algorithms efficiency and/or effectiveness. Each researcher's work is summarized and categorized based on how coevolution was applied to the MOEA algorithm. We also look at some potential future applications of coevolution and describe which situations they might be most beneficial.

1 Introduction

In the real-world, many problems are intractable and they are nearly impossible to solve with a deterministic algorithm in a reasonable amount of time. So researchers often use stochastic algorithms as a means of finding a good solution in a reasonable amount of time. Often, these problems can have more than one objective which can run counter to another. Currently, many researchers rely on multi-objective evolutionary algorithms (MOEAs) as a means for solving these multi-objective problems (MOPs). MOEAs are stochastic algorithms that apply the principles of evolutionary algorithms (EAs) to MOPs. MOPs are n-dimensional problems (where n is the number of objectives) where the landscape is typically very large and produces many possible solutions. As such, there are additional factors that an MOEA needs to address that aren't addressed in single objective problems (preserving good solutions, converging toward the optimal solutions, maintaining diverse solutions, and presenting the decision maker with a reasonable number of solutions).

Recently, coevolutionary techniques have been applied to MOEAs. The resulting coevolutionary MOEAs (CMOEAs) have shown some promise, but there is still a lot of untapped potential. One big advantage of CMOEAs is the ease with which they can be parallelized. CMOEAs, by their nature, can decompose easily into smaller parts that be processed simultaneously. By leveraging this inherent parallelism, the efficiency and effectiveness of CMOEAs can be greatly increased.

Section 2 presents a brief overview of EAs and MOEAs. Section 3 presents a discussion on parallel MOEAs. Section 4 discusses CMOEAs that have been implemented by researchers. Section 5 discusses some possible coevolutionary techniques that can be applied to MOEAs. And Section 6 gives a brief summary of this research.

[*]The views expressed in this article are those of the authors and do not reflect the official policy of the United States Air Force, Department of Defense, or the United States Government.

2 Why MOEAs?

EAs include genetic algorithms (GA), evolution strategies (ES), and evolutionary programming (EP). EAs consist of a class of algorithms that leverage the concepts of genetics to enable the exploration and exploitation of a search landscape. An evolutionary algorithm consists of a collection of individuals (chromosomes) called a population. The chromosomes consist of cells (alleles) in which various data types can reside; binary, integer, real-valued. These chromosome allele values are manipulated by EA operators. These manipulated chromosomes, through a selection operator, form a new generation. Genetic operators such as mutation and recombination are typically used to manipulate the chromosomes. Mutation is the process of inserting new genetic material into the population by modifying allele values, typically in a random fashion. Recombination consists of transferring preexisting genetic material or allele values between two or more individuals (parents) in the population. There are many varieties of genetic operators, each with a different set of parameters that may be modified given a particular EA type [1].

The transition to multi-objective EAs (MOEAs) focuses on the phenotype domain with multiple objectives or fitness functions. The genotype domain is essentially the same with each individual chromosome evaluate for each fitness function. MOEAs allow a decision maker to weight the multiple objectives and decide on a solution based on the importance attached to the objectives. There are different ways a decision maker can weight the objectives. One of the most popular methods is by using a Pareto front. The Pareto front MOEA generates multiple good (non-dominated) solutions and allows the decision maker to choose the one that best solves the problem. MOEAs are great tools because, unlike other methods, they can generate non-continuous and non-uniform Pareto fronts. Additionally, MOEAs can typically handle deception problems well.

Due to a lack of space, MOEAs are not discussed in a lot of detail. For more information on MOEAs, see the book by Coello et. al [2].

3 Parallel MOEAs

Many real world problems are quite complex and can be mapped to NP-Complete problems. Typically, these problems are not pedagogical in nature and an optimal solution, using deterministic algorithms, cannot be found in a reasonable amount of time. To obtain acceptable solutions to these problem, researchers often use two approaches: stochastic algorithms and parallel processing. Of all the stochastic methods, EAs are a popular choice for many researchers. An EA is a stochastic method that is capable of finding very good solutions in a reasonable amount of time. But occasionally, there are can be difficult real-world problems which can take an EA a long time to converge to a solution. In order to increase efficiency, a parallel implementation of the EA may be used. EAs are naturally parallelizable because they maintain a population of solutions [3].

But parallel MOEAs (pMOEAs) are fairly new and very little has been published of them. The creation of a pMOEA is not an easy task. It requires the analysis of parallel paradigms and their associated parameters [4]. This section will briefly discuss some of these paradigms and their importance with respect to the problem.

Since MOEAs are stochastic methods, two critical factors that need to be addressed are the efficiency and effectiveness of the pMOEA. A parallel version of an MOEA must remain as effective as its serial counterpart and it must improve upon the efficiency (i.e. "quickness") as well. A parallel algorithm that is faster than a serial algorithm, but doesn't find as many good solutions, is pointless. Likewise, a parallel algorithm that is just as effective as a serial algorithm, but it doesn't improve the time to find the solutions, goes against the principle of Occams Razor, which is usually stated as: "Of two competing theories or explanations, all other things being equal, the simpler one is to be preferred."

For many MOEAs, the major computational bottleneck occurs in the calculation of complex non-

linear MOP functions, similar to the computational fluid dynamic (CFD) problems. By effectively decomposing and distributing the problem to multiple processors, the problem can be solved more efficiently. But even though an MOEA lends itself well to parallelizing, very little interest has been shown. There are only a small number of MOEA publications that either address or attempt pMOEAs [2].

4 Coevolutionary MOEAs

According to the Merriam-Webster Online Dictionary, coevolution is defined as "evolution involving successive changes in two or more ecologically interdependent species (as of a plant and its pollinators) that affect their interactions ". Coevolution can be either cooperative or competitive. Researchers have been applying coevolutionary techniques to EAs for some time, but recently, these techniques have been introduced into MOEAs. Paredis et. al. wrote a good introduction to coevolution [5].

This section explores the research done in the field of coevolutionary multi-objective algorithms (CMOEAs) and notes how these algorithms have been parallelized.

A Starting Point

Perhaps the most common type of CMOEA uses the strategy of decomposing the chromosome into smaller subcomponents and evolving the these subcomponents in parallel. This idea was introduced by Potter et. al. [6] as a way to coevolve an EA. Potter created a cooperative coevolutionary genetic algorithm (CCGA) that decomposes a problem into subcomponents, called species, that are easy to parallelize. He proposed two algorithms - **CCGA-1** and **CCGA-2**. With CCGA-1 each of the species is coevolved in a round robin fashion using a traditional GA. The fitness of each subpopulation member is obtained by creating a fully instantiated chromosome by combining the subpopulation member with the best individual of the other subpopulations. The result becomes the fitness of the individual. This method of credit assignment has several potential problems, such as undersampling and being too greedy, but it was only created as a starting point to base the effectiveness of further refinements of the algorithm. The results showed that CCGA-1 outperformed the standard GA when the species represented functions that were independent of each other but it performed worse when the function had dependencies.

To overcome this deficiency, a new credit assignment was created for CCGA-2 (see Algorithm 1 for details). In addition to creating an individual as described before, a second individual is created where a random member of each subpopulation is combined with the individual. The best fitness of the two individuals is the one assigned to the child's fitness. This change in credit assignment produced an algorithm that did well when the function was independent or when it had dependencies.

CMOEAs with Subpopulations

Three different researchers have applied this strategy to CMOEAs; Keerativuttitumrong et. al. designed the **Multi-objective Co-operative Co-evolutionary Genetic Algorithm (MOCCGA)** [7], Tan et. al. created the **Distributed Cooperative Coevolutionary Algorithm (DCCEA)** [8], and Iorio et. al. proposed the **Non-dominated Sorting Cooperative Coevolutionary Genetic Algorithm (NSCCGA)** [9]. Keerativuttitumrong et. al. integrated the multi-objective genetic algorithm (MOGA) designed by Fonseca [10] with the CCGA algorithm designed by Potter [6]. MOGA is an MOEA that uses a rank based approach to finding solutions on the Pareto front. The MOCCGA decomposes the problem into subpopulations based on the decision variables, or a part of the problem that requires optimization. Each individual of every subpopulation is ranked with respect to its subpopulation based on its Pareto dominance when it is combined with the best individuals of other subpopulations and they are assigned a fitness value based on their rank in the subpopulation. The MOCCGA is tested against the MOGA using six optimization test cases developed by Zitzler [11]. The test problems represent an array of aspects that an MOEA might be faced with. The author finds that both algorithms do a particularly poor job on the discrete and non-uniform problems. But the

Algorithm 1 CCGA-2

```
1: procedure CCGA-2(𝒩, g, f_k(x⃗))
2:     for Each species s do
3:         ℙ′_s(g) = Randomly initialize population
4:         Evaluate fitness of each individual in ℙ′_s(g)
5:     end for
6:     while Termination condition = false do
7:         g = g + 1
8:         for Each species s do
9:             Select ℙ′_s1(g) from ℙ′_s(g − 1) based on fitness
10:            Select ℙ′_s2(g) from ℙ′_s(g − 1) at random
11:            if ℙ′_s1(g) is most fit then
12:                Let ℙ′_s(g) = ℙ′_s1(g)
13:            else
14:                Let ℙ′_s(g) = ℙ′_s2(g)
15:            end if
16:            Apply genetic operators to ℙ′_s(g)
17:            Evaluate fitness of each individual in ℙ′_s(g)
18:        end for
19:    end while
20: end procedure
```

results show that MOCCGA performs better than the MOGA in all the tests that were run. When the MOCCGA is run in parallel, each species is run on its own computer and they broadcast their best decision variable and the whole decision variable to all other processes. The algorithm is tested with small, medium, and large instances of the problem on 1, 2, 4, and 8 processors and with the MPI broadcast compared to a customized broadcast. It is determined that the customized broadcast works best for the 4 and 8 node cases and the speed up is "quite satisfactory".

The DCCEA created by Tan et. al., is basically an extension of Keerativuttitumrong et. al.'s work [7]. The algorithm uses the CCGA cooperative coevolution scheme [6] and the MOGA ranking scheme [10]. Where this algorithm is particularly different is that it uses a different niching mechanism and an archive for the solutions. The niching scheme is one developed by the author, so it differs from the one used in [7]. The biggest advantage to this niching scheme is that it is adaptive and the user doesn't need to set a niche radius a-priori. As for the archive, it contains all the non-dominated solutions up to a user defined maximum. An extending operator is also employed which clones individuals that have the smallest niche count and assigns the pieces of the clones to the various subpopulations. The hope is that doping the subpopulations can bias the algorithm to areas of the Pareto front that need further exploration. The parallelization of the DCCEA is a coarse-grained strategy, similar to the one employed by Keerativuttitumrong [7], but more involved. The subpopulations are combined into peer groups and each peer group is assigned a computer. These peer groups have their own archive and generate complete solutions. After several generations (an exchange interval), each peer group submits its archive and representatives to the central server and downloads the updates from the other peers. The peers are synchronized according to a user defined synchronization interval to avoid performance deterioration. This distributed algorithm is imbedded into a distributed computing framework called Paladin-DEC [8, 12].

DCCEA is tested using five of the six test functions that Zitzler introduced [11]. The results are compared with five other MOEAs, based on generational distance (how far the known Pareto front is from the true Pareto front) and spacing (how evenly distributed the members are along the known Pareto front) [2]. He finds that with respect to generational distance, his results are comparable to the other MOEAs, but it does especially well on the multi-modal and nonuniform test cases. With regard to spacing, his algorithm consistently had the best spacing.

The NSCCGA proposed by Iorio et. al. combines many of the aspects of the CCGA [6] and the NSGA-II [13] MOEA. The algorithm decomposes the problem into subpopulations based on the number of variables (genes) and coevolves each subpopulation in an attempt to find the true Pareto front for a problem. It forms collaborations in a manner slightly different than [6]. Instead of choosing the best individual from each subpopulation, a random individual is chosen from a group of "best" individuals. Any individual that was a segment of a non-dominated solution is contained in that group. The individuals are ranked based on the NSGA-II Pareto ranking scheme, where all non-dominated individuals are ranked as a one and removed from the population. Then the next set of individuals are ranked as a two, and removed from the population. This continues until all individuals in the population have been ranked. Since each subpopulation holds a static number of individuals the algorithm uses the NSGA-II's crowding distance [13] to determine which individuals to keep from the same rank. The algorithm is compared to the NSGA-II on five problems proposed by Zitzler [11] and a rotated problem proposed by Deb [13]. The NSCCGA performed well compared to the NSGA-II in all cases except the rotated problem. The rotated problem is an example of variables that are not independent of one another. The problem is decomposed in such a way that it is assumed that all the variables are independent. This is a drawback for all subpopulation-based coevolutionary algorithms. While no parallel experiments were run for this algorithm, parallelization of the algorithm would be similar to the previously mentioned algorithms since their structures are similar.

Paremee et. al. created **Parmee's Co-Evolutionary MOEA** [14] which also decomposes the problem, but slightly different than the previously mentioned algorithms. Parmee's CMOEA was used as a preliminary method to identify feasible design regions. It's goal is to initially narrow the search space in the field of airframe design. A population for each objective function is evolved simultaneously. The algorithm utilizes a range constraint map. The results show that the algorithm does a pretty good job at the edges of the Pareto front, but doesn't do as well in the center regions. It also appears that the reduction cycle may play an important role in the solution. Running the algorithm in parallel produced a linear decrease in running time.

CMOEAs and Niching

Two algorithms, the **Elitist Recombinative Multiobjective Genetic Algorithm with Coevolutionary Sharing (ERMOCS)** [15] and the **Genetic Symbosis Algorithm (GSA) for MOPs** [16] apply coevolution to the niching portion of an MOEA. The ERMOCS, developed by Neef et. al., uses a rank-based selection scheme, where an individuals rank is determined by the number of individuals that dominate them. The algorithm uses coevolutionary shared niching (CSN). CSN was first introduced by Goldberg [17] as a way to dynamically change the niche size and location of a population. He does this by using an economic model known as a model of monopolistic competition. This model coevolves customers and businessmen, where a customer is a solution and a businessman is a is the location of a niche in the phenotype space. The niches move dynamically and the fitness is determined by the sum of the rank of the solutions located in the niche radius. So businessmen in sparse regions along the Pareto front are more fit than ones in densely populated regions. A disadvantage of this algorithm is the fact that a minimum distance parameter needs to be set. Several options are presented to alleviate this situation. This algorithm was only run serially, so no parallel information is currently available. But a parallel implementation could easily be added to the algorithm by decomposing the information based on customers or businessmen. These populations could then also be decomposed to smaller populations and parsed out to different machines.

The GSA for MOPs was created by Mao e.t al. as an extension of his single objective GSA [18] to handle multiple objective problems. The GSA portion of the algorithm has two features that are different from the standard GA. The first feature is the introduction of a parameter that attempt to represent the symbiotic relationship between individuals. The second feature is a symbiotic parameter that describes the interaction of the objective functions. This second parameter functions in the same manner as the initial symbiotic parameter. There are five weighting factors that the designer must

plug into the algorithm. These weight the importance of the mean and variance in the phenotype and genotype domains and the weight of the ranks of the solutions. But determining these weights means the user might need to have an idea of what the search space looks like. An earlier paper appears to show that adjusting the settings produces varying results when applied to a single objective [18], so it seems that some knowledge of the search space is beneficial in determining the parameters. The algorithm results are not compared to any other MOEA results, so the effectiveness of the algorithm, with respect to similar methods, is unknown. Additionally, no parallelization efforts were attempted with this algorithm. But parallelization can be obtained by decomposing the population and using an island model.

Other CMOEA Methods

Barbosa et. al. proposed an **interactive genetic algorithm with co-evolving weighting factors (IGACW)** for the objectives along with the population [19]. This algorithm has two populations, one for the proposed solutions to the problem, and the other for the weight-set population. The populations are evolved in a round robin process. Then, after a specified number of generations, a set of solutions are presented to the user for inspection and ranking. The ranking is based on user preference and may be different from the actual fitness assigned to the graphs by the algorithm. The algorithm completes when the user is satisfied with the results. The results show that the evolution process for the weights is greatly affected by the user preferences. The weights associated with the criteria evolved to quite different values in the two runs. The solutions also evolved to greatly different graphs. Since this type of testing is highly subjective on the user's preference, it would be hard, if not impossible, to compare the IGACW algorithm with another GA and obtain objective results. This algorithm was not parallelized, but it could be based on the two populations that are being coevolved, the solution set and the weight set.

Lohn et. al. implemented a version of a Coevolutionary Genetic Algorithm, which is referred to **Lohn's CGA** [20]. This algorithm utilzes two populations, a solution population and a target vector population. Both populations are evolved using genetic algorithms. Individuals in the solution population have their fitness based on how many target objective vectors (TOVs) they successfully solve as well as the difficulty of those target vectors. The individuals in the TOV population have their fitness based how many solutions can achieve their settings. The highest fitness score is achieved when only one solution meets the TOV. The worst fitness score occurs when either all solutions or no solution eclipses the TOV. Lohn's CGA is compared to the results found in [11]. The algorithm is run on the six test functions and then compared to the results of the other MOEAs. The CGA algorithm did well compared to all the MOEAs on the test functions. Only the SPEA was able to find any of the Pareto optimal points from the nonuniform test function, but the CGA provided close results with very good coverage. On the deceptive test function, the SOEA performs better at low f_1 values but the CGA has excellent coverage along the rest of the Pareto front. As far as coverage goes, the CGA does very well compared to the other algorithms. Its weakest performance is on the nonuniform test function when compared to the NSGA, SOEA, and SPEA. This algorithm was not parallelized but it could be decomposed based on the populations. The TOV population could be farmed out to on processor and the solution population could be farmed out to another. Further decomposition of the individual populations could also be implemented.

Coello proposed a **Coevolutionary MOEA** that he labelled the **CO-MOEA** [21]. The CO-MOEA decomposes the problem into competing populations. If a population produces more individuals on the known Pareto front, then it is rewarded by having its population size increased. The goal is to direct the search to the most promising regions of the search space. The algorithm runs in four stages which help to regulate the competing population sizes. The algorithm is compared with the three other MOEAs. Quantitative and qualitative comparisons illustrate how each algorithm does with respect to the other algorithms. He finds that the CO-MOEA is competitive with the other MOEAs. The two disadvantages mentioned in the paper include the high selection pressure induced by the algorithm's elitist scheme,

and the number of populations that the algorithm may need to use. This algorithm was not parallelized, but it could be, by decomposing the population into smaller populations and using an island model.

Table 1 lists the coevolutionary MOEAs discussed in this paper. It labels each algorithm as either symbiotic (host and symbiont populations interact with each other),cooperative (populations cooperate to achieve the best solution), or competitive (populations compete to achieve the best solution). It also shows which algorithms decompose the chromosome into smaller subpopulations. The table also states where coevolution is applied in the various algorithms.

Table 1: **Coevolutionary Techniques Used in MOEAs**

Algorithm	Symbiotic, Cooperative or Competitive	Subpopulations Used	Where applied in algorithm
ERMOCS [15]	Cooperative	No	Niching
Parmee's CMGA [14]	Cooperative	Yes	Population
GSA for MOP [16]	Symbiotic	No	Niching
IGACW [19]	Cooperative	No	Objective Weighting
MOCCGA [7]	Cooperative	Yes	Population
Lohn's CGA [20]	Cooperative	No	Fitness Function
DCCEA [8]	Cooperative	Yes	Population
CO-MOEA [21]	Competitive	No	Population
NSCCGA [9]	Cooperative	Yes	Population

5 The Next Frontier?

Coevolving Multiple MOEAs

A common cliche is "Two heads are better than one". This could also be applied to MOEAs. Suppose we have an MOEA that is particularly effective at finding solutions to deceptive problems and we have another MOEA that is effective at solving non-uniform Pareto fronts. If we don't know the phenotype space of the problem, throwing multiple MOEAs with distinctive strengths may be better than using just one. The MOEAs could be coevolved in such a way that individuals are passed from one algorithm to another in an attempt to seed the algorithms with good solutions.

One may ask how this is different than using a memetic algorithm? Typically, memetic algorithms run the multiple algorithms in a serial fashion or where one algorithm is dominant over another. In this case, the algorithm run in the same manner they would individually. This may prove to be a more efficient way to solve a problem than using either a single MOEA or using a memetic algorithm.

Coevolving MOEAs with other Search Algorithms

MOEAs have been developed which incorporate local search at some point in the algorithm. But an interesting application may be to coevolve an MOEA with another type of algorithm, such as a local search, Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), simulated annealing, etc. By coevolving, the search can exert a blend of exploitation and exploration that occurs in each generation of the algorithm. There are many ways that this coevolution can take place, but this section limits the discussion to one particular implementation.

Suppose you have x populations, one is evolved using an MOEA and the other $x-1$ are local search algorithms. It the first generation, the MOEA runs and ranks the top solutions, using a Pareto ranking system and a crowding mechanism. The top solutions are then used to seed the local search algorithms.

While the local search algorithms are working to find better solutions, the MOEA is also evolving. After a set number of generations, the algorithms cross-pollinate, and they evolve using the new members in their population. A mechanism would have to be devised so that an individual is only submitted to a local search mechanism a set amount of times. This allows for other avenues of exploration in the search. But the populations could be designed so that there are varying degrees in the local search. For example, one local search may flip only one bit or it may focus on only one segment of the chromosome, while another population may flip 4 bits or focus on a separate part of the chromosome.

Coevolving Archives using Multiple Niches

Different algorithms use different types of crowding mechanisms and store their results into an archive. What if the algorithm coevolved multiple archives based on two different niching schemes? Each individual in a population could be labelled in some way so that the algorithm can determine which points are different in the archives. The archives can be combined into a larger population that is evolved. This type of coevolution doesn't appear to have too much value, but it may be able combine various nicning procedures to create a more diverse population than one niching operator can.

Coevolving Target Solutions

Coevolution could also be applied in a manner similar to the one prescribed in by Lohn et. al. [20]. In that research, target objective vectors guide the search toward the Pareto front by increasing the fidelity of the fitness required to be considered a "good" solution. This moving target is an attempt to guide the solutions toward better solutions. This could possibly be done in a manner similar to niching. As a population evolves, the fitness of individuals are based their distribution along the Pareto front. Target vectors are evolved at the same time and these vectors are placed in areas throughout the Pareto front. If many points fall into neighborhood near a target vector, its fitness is low, but if only one or two vectors fall into a target vector neighborhood, it achieves a high fitness score. The target vectors evolve to fill less sparse locations along the Pareto front. In turn, the population of solutions are rewarded with higher fitness values when they are closer to a target vector with only a few points in its neighborhood. This method may prove to be alternative to some of the niching methods currently in use.

Coevolving Competing Populations

In Coello's CO-MOEA [21], populations compete implicitly. A population is rewarded with more individuals when it contributes more to the known Pareto front. But there is no explicit competition between them. A population didn't have a goal to have more individuals than another population, it just happened by chance. But what if there were explicit competition between populations, in which the goal of each population is to gain more individuals? We propose a way that the populations could coevolve and compete in a way that they drive each other to obtain better solutions.

You could set-up an MOEA that has multiple populations, each with the same initial size. Each population's individuals are fully instantiated, meaning an individual is a complete solution to the problem. The populations evolve as any typical GA population does, using selection, mutation, and recombination operators. A global Pareto front is generated from the best individuals from all the populations. The population that generated the most individuals is rewarded with more population members and the population with the fewest members is punished with fewer members. A population that has fewer members adjusts it's operator parameters and quite possibly the operators it uses in an effort to improve its search capability. A population that gains members maintains status quo.

Pitting populations against each other and allowing them to vary their operators and parameters can be very beneficial, especially for unknown search landscapes. This self-preservation mechanism allows the populations to self-adapt and create an algorithm that is better suited to the search space. Plus, if a good population starts to lose individuals, it can adapt in an effort to avoid stagnation. This type of competition has a lot of potential.

6 Conclusion

The CMOEAs reviewed in this paper set the stage for many possibilities in the future. Some future implementations that are currently being studied have been discussed. CMOEAs decompose nicely, so they are can be easily implemented in a parallel fashion to produce better effectiveness and/or efficiency. CMOEAs show a lot of promise and may be able to provide better results when faced with unknown search landscapes.

References

[1] Thomas A. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York - Oxford, 1996.

[2] Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, May 2002. ISBN 0-3064-6762-3.

[3] Larry J. Eshelman. Genetic algorithms. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Evolutionary Computation 1: Basic Algorithms and Operators*, chapter 8, pages 64–80. Institute of Physics Publishing, Bristol, 2000.

[4] David A. Van Veldhuizen, Jesse B. Zydallis, and Gary B. Lamont. Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 7:144–173, April 2003.

[5] Jan Paredis. Coevolutionary algorithms. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Evolutionary Computation 2: Advanced Algorithms and Operators*, pages 224–238. Institute of Physics Publishing, Bristol, UK, 2000.

[6] Mitchell A. Potter and Kenneth A. De Jong. A cooperative coevolutionary approach to function optimization. In *PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, pages 249–257, London, UK, 1994. Springer-Verlag.

[7] Nattavut Keerativuttitumrong, Nachol Chaiyaratana, and Vara Varavithya. Multi-objective cooperative co-evolutionary genetic algorithm. In *PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, pages 288–297, London, UK, 2002. Springer-Verlag.

[8] K.C. Tan, Y.J. Yang, and T.H. Lee. A Distributed Cooperative Coevolutionary Algorithm for Multiobjective Optimization. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 4, pages 2513–2520, Canberra, Australia, December 2003. IEEE Press.

[9] Antony Iorio and Xiaodong Li. A cooperative coevolutionary multiobjective algorithm using non-dominated sorting. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell, editors, *GECCO (1)*, volume 3102 of *Lecture Notes in Computer Science*, pages 537–548. Springer, 2004.

[10] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[11] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[12] Kay Chen Tan, Arthur Tay, and Ji Cai. Design and implementation of a distributed evolutionary computing software. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 33(3):325–338, 2003.

[13] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[14] Ian C. Parmee and Andrew H. Watson. Preliminary airframe design using co-evolutionary multiobjective genetic algorithms. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1657–1665, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[15] Martijn Neef, Dirk Thierens, and Henryk Arciszewski. A case study of a multiobjective recombinative genetic algorithm with coevolutionary sharing. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 796–804, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.

[16] Jiangming Mao, Kotaro Hirasawa, Jinglu Hu, and Junichi Murata. Genetic Symbiosis Algorithm for Multiobjective Optimization Problem. In *Proceedings of the 9th IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN 2000)*, pages 137–142. IEEE, 2000.

[17] David E. Goldberg and Liwei Wang. Adaptive niching via coevolutionary sharing. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 21–38. John Wiley and Sons, Chichester, 1998.

[18] K. Hirasawa, Y. Ishikawa, J. Hu, J. Murata, and J. Mao. Genetic symbiosis algorithm. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 1377–1384, Piscataway, NJ, 2000. IEEE Service Center.

[19] Helio J. C. Barbosa and Andre M. S. Barreto. An interactive genetic algorithm with co-evolution of weights for multiobjective problems. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 203–210, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[20] Jason Lohn, William F. Kraus, and Gary L. Haith. Comparing a coevolutionary genetic algorithm for multiobjective optimization. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1157–1162. IEEE Press, 2002.

[21] Carlos A. Coello Coello and Margarita Reyes Sierra. A Coevolutionary Multi-Objective Evolutionary Algorithm. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 1, pages 482–489, Canberra, Australia, December 2003. IEEE Press.