Late Parallelization and Feedback Approaches for Distributed Computation of Evolutionary Multiobjective Optimization Algorithms

O. Tolga Altinoz
Department of Electrical and Electronics Engineering
Ankara University
Ankara, 06830, Turkey
e-mail: taltinoz@ankara.edu.tr

Kalyanmoy Deb
Department of Electrical and Computer Engineering
Michigan State University
East Lansing, MI 48864, USA
e-mail: kdeb@egr.msu.edu

COIN Report Number 2015016

Abstract—**Distributing of the multiobjective optimization algorithm into various devices in a parallel fashion is a method for speeding up the computation time of the multiobjective evolutionary algorithms (MOEAs). When the processors are increased in number, the gain from parallelization decreases. Therefore, the aim of the parallelization method is not only to decrease the overall algorithm execution time, but also to obtain a higher gain from the use of parallel processors. Therefore, in this study two new parallelization approaches are proposed and discussed, which are named as late parallelization (no-migration approach) and feedback approaches. The performances of these approaches are evaluated on convex and concave multi-objective test problems.**

Keywords-reference point-based NSGA-II; parallelization; distributing computing

I. Introduction

An efficient distribution of a multiobjective optimization task to various devices in a parallel manner is an innovative method for speeding up the computation time of the evolutionary multi-objective algorithms (MOEAs) [1]. Various parallelization methods were proposed to increase the gain from parallel algorithm in the past. These methods can be grouped as (i) master-slave model [2] and (ii) island models [3]. Master-slave model based approaches utilize one master device, which is in charge with the distributing of the parallel modules of algorithm, collect the results, and evaluate them. Generally, objective functions are computed on various parallel devices as application of mater-slave model. The island model is based on information sharing between interconnected free devices [4]. The free device means that in general, these devices can have their own search space, their own population (sub-population) and even their own different algorithms (hybrid implementation). Recently, the authors introduced a new method in which a set of reference points was distributed among parallel MOEAs [5]. The idea is simply based on the distributing the reference points to the different parallel devices where each device is expected to find Pareto-optimal points corresponding to its own reference points. Although the initial results show the efficiency of the parallel performance of the algorithm, still the approach needs further investigation for possible improvement. Also in the same study, a basic method called "delayed approach" was introduced and the performance was improved for a two-processor system with the delayed approach. However, for a three-processor system, the desired linear speedup could not be obtained. Hence in this study, two new parallelization approaches are proposed and discussed for solving multi-objective optimization problems. The aim of this paper is to reach to a linear speedup with an increase in number of processors (up to eight processors are used here). This paper is organized as follows: Section II gives a detail description of two novel proposed methods with a brief explanation of the optimization algorithm. The implementation and their results are divided into two sub-sections based on two approaches at Section III. The last section outlines the conclusion of this study.

II. Proposed Distribution-based Approaches

In this section, two novel parallelization approaches are discussed. After the brief introduction of the optimization algorithm (RNSGA-II) [6] first the late parallelization is explained. This idea is the improved version of the authors' previous idea named as "delayed approach" [5]. Secondly, feedback idea is produced from the late parallelization approach is explained in detail.

A. *Reference Point-based NSGA-II Algorithm*

Reference point based NSGA-II (RNSGA-II) is a modified version of original NSGA-II with the tournament selection, the real-coded SBX recombination, and the polynomial mutation operators [7]. Instead of the crowding distance operator, RNSGA-II considers the normalized Euclidean distance between reference point(s) and population members. The aim of the algorithm is to emphasize non-dominated solutions that are “closest” to the supplied reference point(s). In the event of multiple reference points, since a widely spread out reference points are supplied in the objective space, RNSGA-II is expected to find a widely separated set of Pareto-optimal front at the end. Since, we use RNSGA-II concept in our two proposed approaches, we discuss some details of the RNSGA-II method first. The normalized distance is calculated as follows:

1. First, the combined population (parent and newly created offspring populations) of size $2N$ is classified into a number of non-dominated fronts of increasing domination level, as done in NSGA-II [7].

2. For each non-dominated front starting with the first one, calculate a normalized and weighted Euclidean distance d_{ref},
3. The ε-clearing idea is applied next to maintain a diverse set of solutions near each reference point. First, the closest population member $\boldsymbol{x}_R$ for reference point R on each front is identified and saved as the top-most member of a sorted list. Thereafter, all neighboring front solutions within a normalized Euclidean distance of ε from $\boldsymbol{x}_R$ are temporarily cleared for further consideration. The next closest solution to the same reference point on the front is then identified and declared as the next member on the sorted list. This process is continued until all first front members are considered. Thereafter, the procedure is repeated for the second front members and so on. After considering all fronts, if still the size of the sorted list is less than the population size, the cleared members are considered starting from first front. A similar clearing mechanism is followed while picking solutions. This procedure is stopped as soon as the sorted list has N members. In this study, RNSGA-II algorithm is used as the optimization algorithm for each processor which is assigned a different set of reference points.

B. *Proposed Late Approach*

First, since it is the starting point of this study, the delayed approach is explained briefly. The delayed idea was proposed by the authors of this study [5]. The idea is simply based on initially starting the algorithm with a single processor. Then, after a specific number of generations, the population is divided into sub-populations (sorting the population based on the division axis and equally dividing the population into processors) and distributed to the parallel processors. This no-migration delay idea increases the performance of two parallel systems such that the total number of function evaluation of the parallel system is same or smaller than single processor performance by slightly sacrificing the parallelization advantage. In spite of this performance on two-processor systems, the performance of delayed idea on three-processor systems could not reach that level. Beside sensitivity to the number of processors, deciding the distribution time (delayed generation t_{delay}) is another issue with the approach. Therefore in this study two novel ideas are proposed and one of them named as late approach, which is introduced to eliminate the above-mentioned difficulties. In this study, a parallel system with a maximum of eight processors is investigated as a difficult test case. The late parallelization approach is a layered version of the delayed idea such that instead of single processor initial run; the single and parallel runs are executed initially. This idea demonstrated in Figure 1. Four-layered structure is demonstrated and implemented in this study. Although, it is easily changed to different layers and number of processors, the algorithm first begins with a single processor and N population members created at random. After a certain number of generations the population is divided into two processors. In other words, the algorithm moves from layer 1 to layer 2. This process is repeated until the last layer (with eight processors) is reached. The idea behind this approach is to make a more global search initially and then focus to multiple distributed local searches with generations. By this way as the overall population begins to converge to the Pareto-optimal front, it begins to search more local areas with a parallel system having two, four or eight processors. As shown in Figure 1, the number of layers and the number of processors in each layer can be changed.

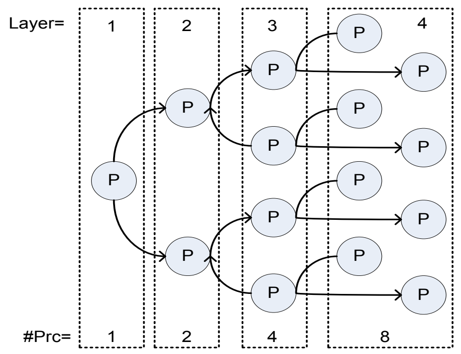

Figure 1. Graphical demonstration of the proposed late parallelization idea where #Prc corresponds to number of processor (P) at each layer.

C. *Feedback Approach*

The feedback approach is an improved version of the late parallelization idea in that instead of a single flow among the layers, the feedback connection is added to the parallelization system as demonstrated in Figure 2. In late parallelization, once the algorithm reaches to the last layer, the optimization processor continues on this process with the last layer configuration. Therefore, there isn't any migration (information sharing) between processors. However, with the new connection between different layers, the information of each processor is shared among other processors. For example, for a connection given in Figure 2a, the implementation begins as explained in the late parallelization approach. After the last layer is reached and finished, the sub-populations are collected and then a single processor optimization is run. Then, as explained in the previous section, it continues as layer 2, layer 3 and layer 4. All processes are repeated until the algorithm termination condition is reached. Figure 2 gives some of the other possible feedback approaches, which are investigated in this paper. In Figure 2b, a loop between layer 1 and layer 4 is defined. Layer 1 and layer 4 are repeated until the end of the algorithm. In other words, after the eight-processor system, all sub-populations are collected and evaluated on single processor. That causes global and local searchers to be executed in a repeated manner. Figure 2c is a kind of hybrid connection of previous two cases (Figure 2a and Figure 2b). Initially all layers are run respectively. After the last layer is evaluated, a loop between layer 1 and layer 4 is executed as explained for configuration in Figure 2b. Figure 2d shows a similar connection as the one in Figure 2c. The only difference is that initially, instead of all layers, only layer 1 is executed three times. The last configuration given in Figure 2d is the inverse connection version of the configuration presented in Figure 2c. Instead of beginning with the single processor, the algorithm begins with eight independent processors (mimicking an initial distributed local search), and then at each generation, all sub-populations are collected and allocated to a fewer processors. In this study, these five different feedback connections are evaluated and compared with a single processor implementation. The aim of these two new parallelization approaches is to reach almost linear speedup [8,9] for a eight-processor system (relatively large number of processors) by sacrificing the full parallelization advantage.

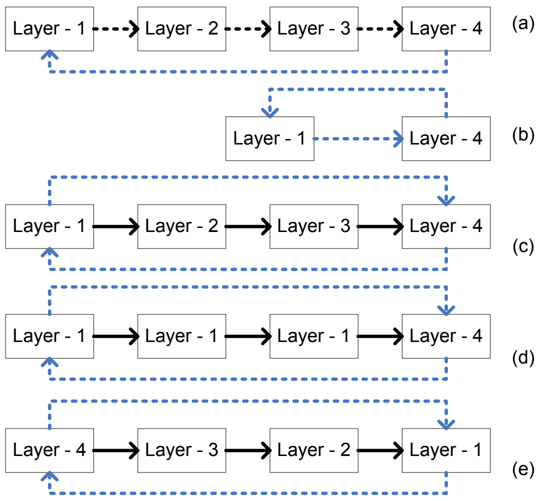

Figure 2. Graphical demonstration of the feedback parallelization idea on different connection

III. Implementation and Results

In this study, RNSGA-II algorithm is used as optimization algorithm to solve two (convex and concave) problems: ZDT1 and ZDT2. The mathematical equation of these test problems can be found in [9]. The population size is chosen to be 160 for single processor and implementations are repeated for 10 independent runs. The statistics of the total number of function evaluations are collected as their minimum, maximum, mean and standard deviation. The termination condition of each implementation is restricted with the hyper-volume indicator such that a run is terminated when the hyper-volume metric is greater or equal to 0.794 with hyper-volume reference point is (1.0646,1.0646) for ZDT1 problem and 0.46 for the ZDT2 problem with the same reference hyper-volume point. At first, late parallelization approach is implementation with respect to various layer transition generation sizes. Then, feedback is added to the late parallelization system and different connections are tried to obtain better performances. The average of the total number of function evaluations at single processor 6,592 and 7,928 with average generations of 40.2 and 48.8 for ZDT1 and ZDT2, respectively.

A. *Layer Approach Implementations*

The idea is graphically demonstrated at the beginning of the paper in Figure 1. In summary, layers are evaluated concurrently witout any relation between each other. The implementation begins with the single processor. After some generations, the population is sorted based on the first objective value and divided into the same number of sub-populations, which are distributed to two processors. This process is repeated until layer 4 (eight processors) is executed. Hence the number of transitional generation size is important and must be determined accurately. In other words, after a specific number of generations, the algorithm switches from layer n to layer $n+1$ $(n=1,2,3)$, where t_n is defined as an indicator of transitional generation size at layer n. At the beginning, each considered to be the same ($t_1=t_2=t_3$). Therefore the number of generations between each layers is selected in [1, 10]. Because for a higher interval value, it may not possible to reach the final layer with an identical number of function evaluations as it is for the single-processor runs. Figure 3 shows the results for the equal transitional generation time for ZDT1 and ZDT2 problems.

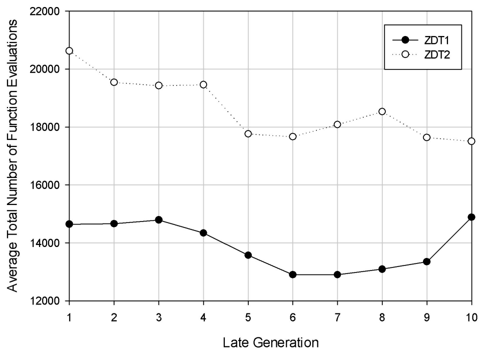

Figure 3. Equal size transitional generation size performance with respect to the average of the total number of function evaluations.

Figure 3 gives the performance of the late parallelization approach on ZDT1 and ZDT2 problems for different late generation times (x axis). The results show that a transition after every six generations is the best choice. The best function evaluation sizes are 12,896, 17,664 and the average generational sizes are 80.6, 110.4 for ZDT1 and ZDT2, respectively. When compared to single-processor performance (65,92 and 7,928), the proposed approach needs 1.95 and 2.22 times more function evaluations. Eight processors evaluate also almost 78% and 84% of average generations. In addition almost 95% of the generation is calculated on parallel devices. Even the parallelization is almost preserved for this implementation; the total number of FEs is still very high compared to that for a single processor. Next, linearly decreasing and increasing generational size is defined for parallelization. Two cases are defined for this purpose. Case 1 is defined as follows. For each generation beginning from the top, generational gap is set as k, $2k$ and $4k$ ($t_1=k$, $t_2=2k$, $t_3=4k$). For example, for $k=2$, layer 1 is run for two generations, layer 2 is for four generations and layer 3 is run for eight generations. Inversely, Case 2 is defined as follows: $4k$, $2k$ and k ($t_1=4k$, $t_2=2k$, $t_3=k$). Figure 4 gives the performance for different k in [1, 5], since $k=5$ correspond to 35 (smaller than single processor average performance) generations without eight processors. Figure 4 shows that the results are not able to show a consistency for both cases. Different k values are obtained for different cases and also for different problems. For Case 1, the best performance is obtained for ZDT1 and ZDT2 as 12,672 ($k=3$), 17,568 ($k=4$) average total FEs and 79.2, 109.8 as average generations, respectively. Almost the same performance is reached when compared to the previous implementation (t_1=t_2=t_3). The only difference is that instead of almost 18 generations required in the previous implementation, 21 (74% of all generations are required by eight processors) and 28 (75% of all generations are required by eight processors) generations are evaluated by single, two and four processors (but, not eight processors) for the best performance. For Case 2, the best performance is obtained for ZDT1 and ZDT2 with 131,84 (*k*=3), 16,000 (*k*=5) average FEs and in 82.4, 100 average generations, respectively. In addition, 21 (75% of all generations are required by eight processors) and 35 (65% of all generations are required by eight processors) generations are evaluated by single, two and four processors. All results in this section show higher number of function evaluations when compared to the single-processor performance. Among all implementations, equally divided late generational time gives the overall best and consistent performance for both problems for a relatively higher parallelization task.

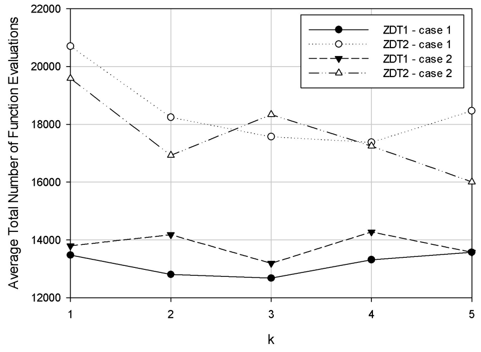

Figure 4. Linearly decreasing (case 1) and increasing (case 2) transition generation size performance with respect to the average of the total number of function evaluations.

B. *Feedbck Implementations*

In this section, we investigate the effect of feedback parallelization process. As explained in late parallelization process, same steps are evaluated in this new approach as well; the only difference is in forming a repeated loop with a feedback. Although many different processors and many different connections can be defined for feedback implementation, only five cases are considered here as shown in Figure 2. The aim is to find the best combination to find the same or better average total FEs than the single processor. At first, two connections (Figure 2a and Figure 2b) are considered. Figure 2a shows the most basic form of the connection such that each layer is connected to each other and formed a loop such at after the last layer the evolution continues from the first layer, and this evaluation is repeated. Here, we only use one generation for each layer, but is also possible to consider more generations at the transient between layers. However, from the results of late parallelization, in this section, equal generational gaps are considered and selected as one. Figure 2b shows the most basic form of the loop connection such that single-processor and eight-processor system are connected to each other. Therefore at every generation these two systems are evaluated in turn. In other words, each sub-population is gathered and formed the overall population. Then this new population is applied to RNSGA-II operators, causing to share information (like a migration approach) and obtain a faster convergence to the Pareto-optimal front. Single processor/population reduces the local search ability of the sub-populations on a narrow space assigned with reference points.

TABLE I. PERFORMANCE OF THE FEEDBACK APPROACH ON FIRST TWO CONNECTIONS IN FIGURE 2

ZDT1	Single Proc.		Figure2a		Figure2b	
	Gen.	FEs.	Gen.	FEs	Gen.	FEs
Min.	36	5920	37	6080	37	6080
Max.	44	7200	49	8000	45	7360
Mean	40.2	6592	42.4	6944	42.3	6928
Std.Dev.	2.39	383.11	3.3	529.04	2.54	406.52
ZDT2	Gen.	FEs.	Gen.	FEs	Gen.	FEs
Min.	40	**6560**	67	10880	46	7520
Max.	74	12000	77	12480	79	12800
Mean	48.8	7928	71.9	11664	53	8640
Std.Dev.	9.65	1545.37	3.72	596.04	10.5	1680

Table 1 gives the performance of the first two configurations (Figure 2a and Figure 2b). For the first case, since each layer is evaluated only for one generation each time causes almost 25% parallelization with respect to the eight processors. For this small parallelization, still the total number of FEs is larger than the single processor, which is not desired. For the second configuration results, only single processor and eight-processor system is evaluated, which also means that eight processors evaluate almost 50% of the generations. Secondly, the remaining three connections are evaluated and the performances of these connections are reported on Table 2. These three connections (Figure 2c, Figure 2d, Figure 2e) are similar to each other; in addition, the last connection (Figure 2e) is the inverse of Figure2c. The first one (Figure 2c) is initially evaluated for all four layers only one time, than layer 1 and layer 4 is repeated (and forms a loop) as explained for Figure 2b. The second connection (Figure 2d) is similar to the first one (Figure 2c) and the only difference is that, instead of the first four layers, the first layer is evaluated three times. In other words, we over-emphasize global search in the beginning followed by a brief and distributed local search. The last connection is the inverse version of Figure 2c in which the implementation begins with the eight processors, then reducing to layer 3, layer 2 and eventually to a single-processor run. The loop is formed between Layers 4 and 1. Table 2 gives the last three connection performances. These three implementations show almost similar (better for ZDT1 and ZDT2 problems) performance to that of a single-processor run. For both the above implementations, eight processors are evaluated almost 45% of the total generations for all problems. The best performance is achieved with the last connection (Figure 2e) for both ZDT1 and ZDT2 problems even it is possible to obtain better performance with just the ZDT1 problem. But overall, the last implementation produces a better performance.

TABLE II. PERFORMANCE OF THE FEEDBACK APPROACH ON LAST TWO CONNECTIONS IN FIGURE 2

ZDT1	Figure2c		Figure2d		Figure2e	
	Gen.	FEs	Gen.	FEs	Gen.	FEs
Min.	38	6240	33	**5760**	38	6240
Max.	45	7360	38	**6560**	41	6720
Mean	40.6	6656	36.2	**6272**	39	6400
Std.Dev.	2.36	378.62	1.92	307.76	1.41	226.27
ZDT2	Gen.	FEs	Gen.	FEs	Gen.	FEs
Min.	43	7040	46	7840	45	7360
Max.	57	9280	55	9280	51	**8320**
Mean	48.9	7984	49	8320	48.2	**7872**
Std.Dev.	4.25	680.70	4.24	678.82	2.16	346.87

The last connection gives 192 and 56 less average function evaluations than even the single-processor case for ZDT1 and ZDT2 problems, respectively. Also for the configuration in Figure 2d, it is possible to obtain better performance for ZDT1. But for this configuration, ZDT2 required more function evaluations than the single processor run. Overall, the last implementation seems to provide the best performance for these two test problems.

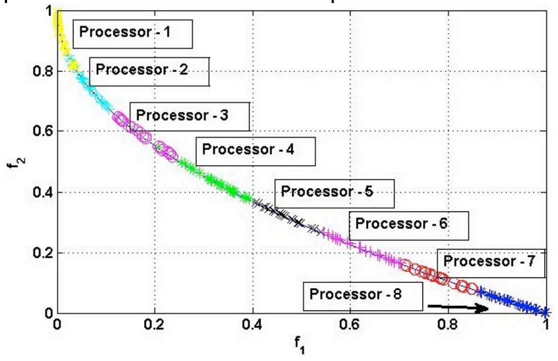

Figure 5. Distribution of the solutionson Pareto front for ZDT1 Problem.

Figures 5 and 6 present the overall distribution of the final Pareto-optimal solutions for the problems ZDT1 and ZDT2, respectively.

IV. CONCLUSION

In this study, two novel parallelization approaches have been introduced and discussed. The first late parallelization approach presents a modified delayed approach. But the performance of the overall parallel system is worse than a single-processor system and needs about two times more function evaluations. In the feedback parallelization approach, the layered parallel structures are connected to another layer and formed a loop. Although a heterogeneous parallel system decreases the parallelization advantage, the overall performance has been found to be better than a single-processor system, which is desired. Each of the feedback approach allowed different combinations of global and a distributed local search. In this initial study, we are able to report a computationally faster approach by using one to eight processors to arrive at a similar performance than that of a single-processor system. The best strategy has turned to be one in which the search is started from a distributed local search in which both diversity and convergence have been paid attention, followed by a more global approach allowing a competition between near Pareto-optimal points for their fine-tuning to secure a place on the final non-dominated front.

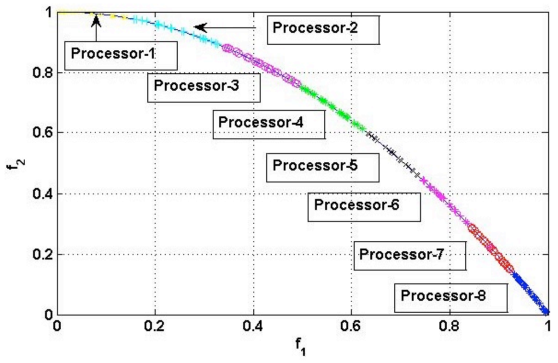

Figure 6. Distribution of the solutionson Pareto front for ZDT2 Problem.

ACKNOWLEDGMENT

O. Tolga Altinoz has been supported by the grant from TUBITAK under 2214/A Research Scholar Program for the research period at Michigan State University.

REFERENCES

[1] E. Alba and M. Tomassini, Parallelism and Evolutionary Algorithms, *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443-462, 2002.

[2] S. Mostaghim, J. Branke, A. Lewis, and H. Schmeck, “Parallel multi-objective optimization using master-slave model on heterogeneous resources,” *IEEE World Congress on Evolutionary Computation CEC-2008*, pp. 1981-1987, 2008.

[3] K. Deb, P. Zope, and A. Jain, “Distributed computing of Pareto-optimal solutions with evolutionary algorithms,” In: C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization*, vol. 2632 of LNCS, pp. 534-549, Springer,2003

[4] J. Branke, H. Schmeck, K. Deb, and S.M. Reddy, Parallelizing multiobjective evolutionary algorithms: cone separation, *IEEE World Congress on Evolutionary Computation*, pp. 1952-1957, 2004.

[5] O.T. Altinoz, K. Deb, and A.E. Yilmaz, Reference Point based Distributed Computing for Multiobjective Optimization, *IEEE World Congress on Evolutionary Computation*, pp.xx-xx, 2015.

[6] K. Deb, J. Sundar, U. B. Rao N, and S. Chaudhuri, “Reference point based multi-objective optimization using evolutionary algorithm,” *International Journal of Computational Intelligence Research*, pp. 273-286, 2006.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, 2002.

[8] H. Schmeck, U. Kohlmorgen, and J. Branke, Parallel implementations of evolutionary algorithms, In: A. Zomaya, F. Ercal, and S. Olariu, editors, *Solutions to Parallel and Distributed Computing Problems*, pp. 47-66, Wiley, 2001.

[9] E. Zitzler, K. Deb, and L. Thieler, Comparison of multiobjective evolutionary algorithms: Empirical results, *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 173-195, 2000