Parallelizing Multi-Objective Evolutionary Algorithms: Cone Separation

Jürgen Branke, Hartmut Schmeck
Institute AIFB
University of Karlsruhe, Germany
Email: {branke|schmeck}@aifb.uni-karlsruhe.de

Kalyanmoy Deb, Maheshwar Reddy.S
Department of Mechanical Engineering
IIT Kanpur, India
Email: deb@iitk.ac.in

KanGAL Report Number 2004017

***Abstract*— Evolutionary multi-objective optimization (EMO) may be computatinally quite demanding, because instead of searching for a single optimum, one generally wishes to find the whole front of Pareto-optimal solutions. For that reason, parallelizing EMO is an important issue. Since we are looking for a number of Pareto-optimal solutions with different trade-offs between the objectives, it seems natural to assign different parts of the search space to different processors. In this paper, we propose the idea of cone separation which is used to divide up the search space by adding explicit constraints for each process. We show that the approach is more efficient than simple parallelization schemes, and that it also works on problems with a non-convex Pareto-optimal front.**

I. Introduction

One possible drawback of evolutionary algorithms (EAs) is that they usually have to evaluate a relatively large number of solutions before generating good results. That is particularly true for multi-objective problems in higher dimensions, as here a rather large population size is required. Luckily, the aforementioned potential drawback is at least partially compensated by the apparent ease of parallelizing EAs. So it comes at no surprise that there is a significant amount of literature on parallelizing EAs. With regards to parallelization, the main difference between single- and multi-objective evolutionary algorithms seems to be that in the multi-objective case, a set of solutions is sought rather than a single optimum. This opens the possibility of having the different processors search for different solutions, rather than to follow an identical goal. The hope is that such a "divide-and-conquer" principle is more efficient that if all processors work on the whole problem. One first approach in that direction has recently been proposed by Deb et al. [9]. That paper was based on an island model with migration and used the guided dominance principle [3] to give the different islands (=processors) different search directions. The approach produced excellent results, in the sense that it converged much quicker to the Pareto-optimal front than without guidance scheme (i.e. when every island searched for the whole Pareto-optimal front). The drawbacks of the approach were that it may be difficult to define appropriate search directions before the shape of the Pareto-optimal front is known. Furthermore, the approach only works if the Pareto-optimal front is convex.

In this paper, we propose an alternative way to allow different processors to focus on different areas of the Pareto-optimal front. The idea is to explicitly divide up the search space into different regions, and to assign each region to one of the processors for exploration.

The paper is structured as follows: In the next section, we will briefly survey related work. Then, in Section III we will present our new approach of dividing up the search space. The approach is evaluated empirically in Section IV. The paper concludes with a summary and some ideas for future work in Section VI

II. Related Work

A. Parallel Evolutionary Algorithms

Evolutionary algorithms are very suitable for parallelization, as crossover, mutation, and in particular the time-consuming evaluation can be performed independently on different individuals. The main problem is the selection operator, where global information is required to determine the relative performance of an individual with respect to all others in the current population. There is a vast amount of literature on how to parallelize EAs. The approaches can be grouped into three categories:

1) Master-slave: Here, a single processor maintains control over selection, and uses the other processors only for crossover, mutation and evaluation of individuals. However, the algorithm is useful only for very few processors and very large evaluation times, as otherwise the strong communication overhead outweighs the benefits from parallelization.
2) Island model: In this model, every processor runs an independent EA, using a separate sub-population. The processors cooperate by regularly exchanging *migrants* (good individuals). The island model is particularly suitable for computer clusters, as communication is limited.
3) Diffusion model: Here, the individuals are spatially arranged, and mate with other individuals from the local neighborhood. When parallelized, there is a lot of inter-processor communication (as every individual has to communicate with its neighbors in every iteration), but

the communication is only local. Thus this paradigm is particularly suitable for massively parallel computers with a fast local intercommunication network.

A detailed discussion of parallelization approaches is out of the scope of this paper. The interested reader is referred to [14], [4], [1].

B. Parallel Multi-Objective Evolutionary Algorithms

Since multi-objective EAs not only search for a single optimum, but usually for a whole set of Pareto-optimal solutions, they are even more in need of efficient parallelization than their single-objective counterpart. Nevertheless, the amount of papers on that topic is limited, for a general discussion see [16].

Several people simply use the master-slave idea, which is straightforward (see e.g. [11]). However, due to its low communication overhead, the island model seems to be the most appropriate parallelization scheme for todays predominant parallel computer architecture, which is simply a cluster of PCs. The island model has been used e.g. in [10], [13]

As has already been mentioned in the introduction, we think that the search for a set of solutions opens the possibility of dividing up the search, and having different islands/processors search for different parts of the Pareto-optimal front. To our knowledge, there are only two previous publications which exploit similar ideas. In the approach by Hiroyasu et al. [15], at regular intervals, the population is gathered, sorted according to one of the objectives (objective used for sorting is chosen in turn), and then distributed onto the different processors. In other words, this approach divides up the population, and has each sub-population work on its share of "similar" individuals with respect to the chosen sorting objective. This implicitly also results in a division of the search space. However, the separation is somewhat arbitrary (according to changing objectives), only temporary (as there is no mechanism to keep populations separated), and the regular collection of individuals requires a lot of communication. An almost identical idea has also been suggested in [5]. The second approach is the aforementioned one by Deb et al. [9]. That paper does not explicitly divide up the search space (all islands search on the whole space), but instead uses the dominance principle [3] to give the different islands (=processors) different search directions. The approach produced excellent results, in the sense that it converged much quicker to the Pareto-optimal front than without guidance scheme (i.e. when every island searched for the whole Pareto-optimal front). The drawbacks of the approach were that it may be difficult to define appropriate search directions before the shape of the Pareto-optimal front is known. Furthermore, the approach only works if the Pareto-optimal front is convex.

III. Cone Separation

As has already been stated in the introduction, the basic idea of this paper is to divide the search space into several regions, which are then assigned to the different processors. Since the fitness space to be searched and the shape of the Pareto-optimal front are usually unknown at the start of the optimization, such a partitioning of the fitness space has to be adaptive. Let us first consider the case of only two objectives being minimized. An extension to three dimensions will be discussed later.

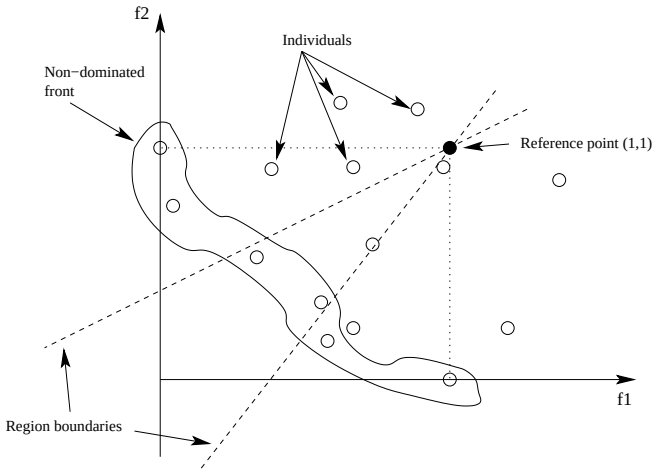

Fig. 1. Example for the partitioning of the (normalized) search space.

The partitioning of the search space is adapted at regular intervals by normalizing the fitness values in such a way that the whole non-dominated front is within the unit square (hypercube in more than 2 dimensions). After the normalization, the fitness space is partitioned into cones by, starting from the reference point (1,1), dividing the $90°$ angle encompassing the non-dominated front into equal parts. Each processor is then assigned one part. Figure 1 illustrates the concept. Note that each region is bounded by two lines, except the leftmost and rightmost regions which are constrained in only one direction.

In order to have each island focus on the specified region, the borders of the regions are treated as constraints, and handled by using the constrained domination principle as suggested in [7], which basically means that all solutions outsided the designated region are dominated by all solutions within.

Frequent normalization adapts the regions to the current search progress. However, re-normalization may cause some very good individuals to violate the constraints of the sub-population they are in. Therefore, whenever the constraints are adapted, individuals violating the constraints are migrated into the population where they do not violate the constraints. Thereby, individuals are just added to the receiving population, without explicitly deleting others (this is done anyway at a later step).

Note that little inter-processor communication is necessary to compute the reference point, since we only need the extreme individuals, i.e. the individuals with minimum f_1 and minimum f_2.

Overall, the approach is integrated into NSGA-II [7] and works as described in Algorithm 1 (steps requiring inter-processor communication are *italic*).

If three objectives are involved, the region constraints are

Algorithm 1 Cone-separated NSGA-II

```
Initialize the different sub-populations
Normalize fitness values
Determine region constraints
Non-dominated sorting
REPEAT
    Generate Offspring
    IF (migration)
        Normalize fitness values
        Determine region constraints
        Migrate individuals violating constraints
    Non-dominated sorting
    Prune population to original size
UNTIL stop-condition
```

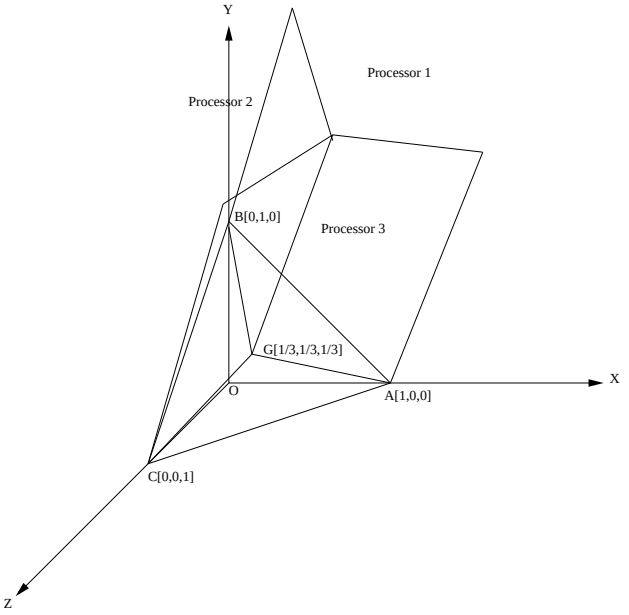

Fig. 2. Example for the partitioning of the search space in 3 dimensions.

determined as follows: first, we determine the plane which goes through each axis at length 1. Then, we simply divide that plane into equal angles, starting at the center $(1/3, 1/3, 1/3)$. Figure 2 illustrates the concept for three processors.

IV. Empirical results

In this section, we test the proposed cone separation approach on a number of different test problems taken from [17] and [8]. We show that the approach is capable of assigning different parts of the Pareto-optimal front to different processors, on a number of test problems including convex, concave, 2D, and 3D Pareto optimal fronts. Furthermore, we show that using cone separation results in faster convergence compared to a simple island model where each population searches for the whole Pareto-optimal front.

As has been described in the previous section, our algorithm is based on NSGA-II. Unless specified otherwise, the standard settings in the experiments reported below were crowded

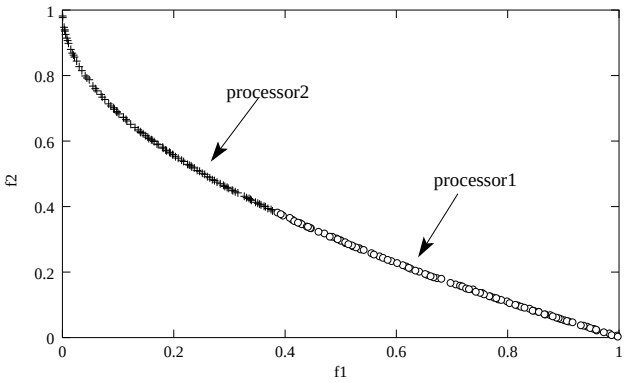

Fig. 3. Non-dominated front obtained with two processors for ZDT1 problem, when hypervolume of 0.794 has been reached.

tournament selection [7], simulated binary crossover [6] with $\eta_c = 10$ and probability $p_c = 0.9$, and polynomial mutation [7] with $\eta_m = 50$ and probability $1/10$. Total population size has been set to 200, standard migration interval is 1, i.e. infeasible individuals migrate in every generation. The algorithm has been implemented in C using MPI[12] for communication between processors. For computation, we use a cluster of Linux PCs connected via Ethernet.

A. ZDT1 problem

Let us start with a very simple multi-objective problem with convex Pareto-optimal front, the 30-dimensional ZDT1:

$$\begin{aligned} \min f_1(x) &= x_1 \\ \min f_2(x) &= g(x)[1 - \sqrt{(f_1/g(x))}] \\ g(x) &= 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\ 0 \leq x_i &\leq 1 \qquad i = 1 \ldots 30 \end{aligned}$$

Figure 3 shows the Pareto optimal front obtained by the two processor cone separation idea, each of the two sub-populations now has 100 individuals. As can be seen, the workload is distributed evenly onto the two processors, and a good overall distribution of individuals on the Pareto optimal front is achieved. Similar results are obtained in the case of 3 and 6 processors with 66 and 34 individuals per island, respectively (cf. Figures 4 and 5).

In the introduction, we conjectured that dividing up the search space should increase the search efficiency In order to validate this conjecture, we will now look at the number of generations an algorithm requires to achieve a given hypervolume[1].

We compare our new cone-separated NSGA-II with

1) The standard single-population NSGA-II
2) A parallel version with an independent NSGA-II running on each processor

[1]The *hypervolume* measures the area dominated by the combined set of all the solutions in the population. It is one of the typical performance measures for multi-objective optimization algorithms (see e.g. [18]).

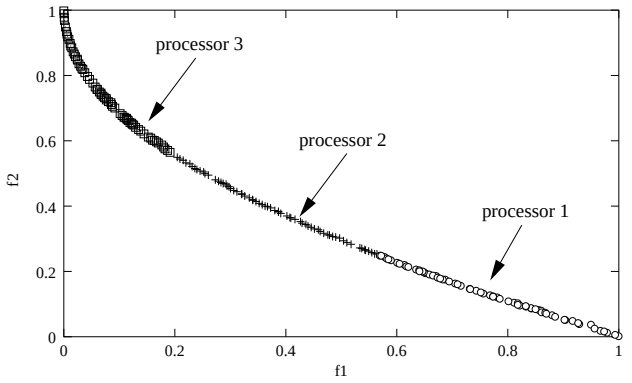

Fig. 4. Non-dominated front obtained with three processors for ZDT1 problem, when hypervolume of 0.794 has been reached.

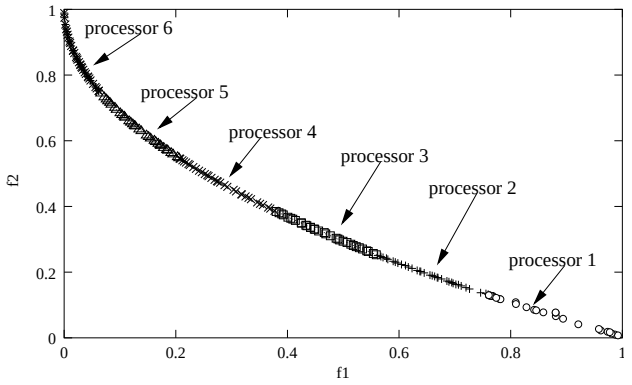

Fig. 5. Non-dominated front obtained with six processors for ZDT1 problem, when hypervolume of 0.794 has been reached.

TABLE I

NUMBER OF GENERATIONS REQUIRED TO OBTAIN HYPER-VOLUME OF 0.794, RESULTS ± STD. ERROR.

Approach	# processors	Popsize/processor	Generations
Standard no migration	1	200	42.5± 2.02
	2	100	54.3± 1.12
	3	66	65.4± 0.70
	5	40	97.2±1.17
Island with migration	2	100	50.0± 2.04
	3	66	60.1± 1.71
	5	40	69.2±0.98
Cone-separated with migration	2	100	41.6± 1.17
	3	66	51.2± 3.36
	5	40	66.9±4.49

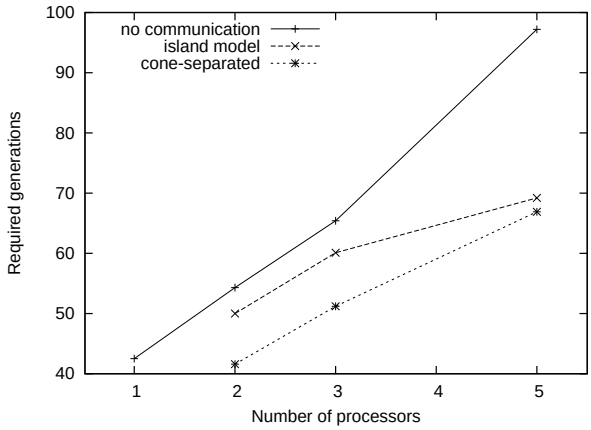

Fig. 6. Required number of generations depending on the parallelization approach and the number of processors used.

3) The standard island model with migration. We tested a number of configurations and here report only on the best configuration, which was to have the two best individuals migrating every second generation.

The results (averaged over 10 independent runs) are summarized in Table I. Figure 6 visualizes the most important details. Clearly, performance deteriorates with an increasing number of processors, independent of the approach used. Simply running several NSGA-II in parallel performs worst. Migration clearly helps, as the island model performs much better. The new cone-separated NSGA-II performs best, confirming our assumption that dividing up the search space and having different populations focus on different areas improves efficiency.

B. ZDT2 problem

Let us now apply the cone-separated NSGA-II to the 30 variable ZDT2 test problem. The Pareto-optimal front of that problem is concave, and could thus not be solved by the guided-domination approach proposed in [9]. ZDT2 is defined as follows:

$$\begin{aligned} \min f_1(x) &= x_1 \\ \min f_2(x) &= g(x)[1-(f_1(x)/g(x))^2] \\ g(x) &= 1+\frac{9}{n-1}\sum_{i=2}^{n} x_i \\ 0 \le\ & x_i \le 1 \qquad i = 1 \ldots 30 \end{aligned}$$

Figures 7 and 8 show the results obtained by the cone-separated NSGA-II on two and five processors, respectively. As can be seen, the cone separation idea works for the concave problems just as well as it did on the convex ones.

C. ZDT3 problem

Now let us look at a problem where the front is not only concave, but also not continuous. ZDT3 is defined as follows:

$$\begin{aligned} \min f_1(x) &= x_1 \\ \min f_2(x) &= g(x)[1-\sqrt{f_1(x)/g(x)} \\ & \quad -(f_1(x)/g(x))\sin(10\pi f_1(x))] \\ g(x) &= 1+\frac{9}{n-1}\sum_{i=2}^{n} x_i \\ 0 \le\ & x_i \le 1 \qquad i = 1 \ldots 30 \end{aligned}$$

The difficulty here is that a particular processor may not know that a solution does not belong to the overall non-dominated front if it only looks at its restricted part of the search space. In order to alleviate this problem, we decided take the extreme individuals from the neighboring processors, and include them in a population's non-dominated sorting. Although this means limited additional communication for

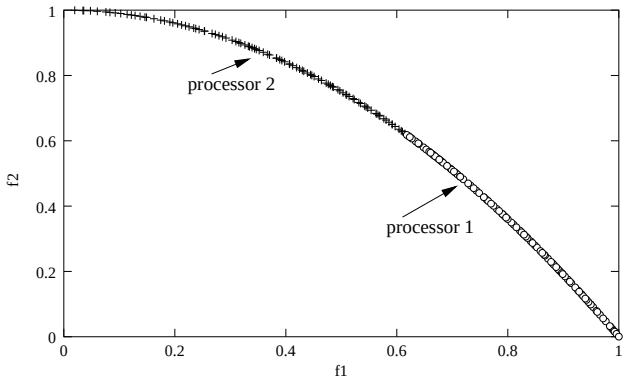

Fig. 7. Non-dominated solutions obtained with two processors for ZDT2 problem.

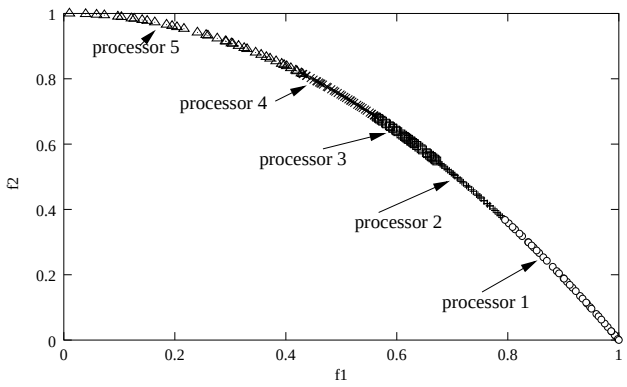

Fig. 8. Non-dominated solutions obtained with five processors for ZDT2 problem.

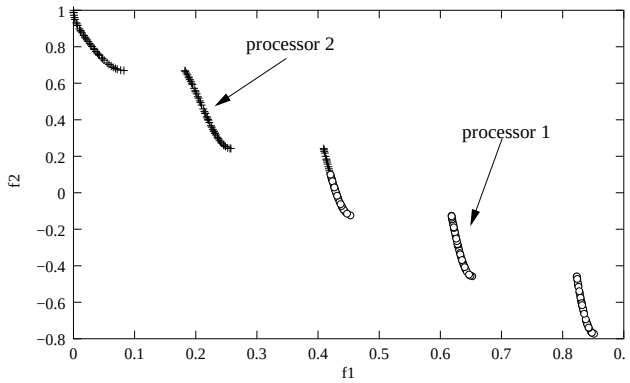

Fig. 9. Non-dominated front obtained with two processors for ZDT3 problem.

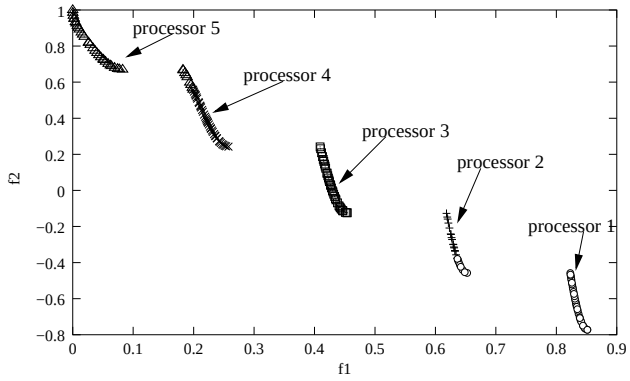

Fig. 10. Non-dominated front obtained with five processors for ZDT3 problem.

exchanging extreme individuals, it seems necessary to obtain satisfying results on this problem.

Figures 9 and 10 again show the resulting non-dominated front generated with 2 and 5 processors, respectively. Again, a good distribution of individuals on the front is obtained.

V. Extension to 3-Dimensions

In this section, we would like to explore the cone-separation idea with 3 objectives. As test problem, we use the modified DTLZ2 problem having $n = 12$ variables:

$$\begin{aligned}
\min f_1(x) &= 2 - (1 - g(\mathbf{x}))\cos(x_1\pi/2)\cos(x_2\pi/2) \\
\min f_2(x) &= 2 - (1 - g(\mathbf{x}))\cos(x_1\pi/2)\sin(x_2\pi/2) \\
\min f_3(x) &= 2 - (1 - g(\mathbf{x}))\sin(x_1\pi/2) \\
g(\mathbf{x}) &= \frac{1}{n-2}\sum_{x_i=3}^{12}\left(\frac{x_i - 0.5}{0.5}\right)^2 \\
0 \leq x_i &\leq 1 \qquad i = 1 \ldots 12
\end{aligned}$$

While we have used a total population size of 200 in the 2D test problems, we are using a population size of 300 for DTLZ2. Figures 11 to 13 show the resulting non-dominated front obtained with two, three or six processors, respectively. As can be seen, in 3D, although the overall distribution is still acceptable, there is an accumulation of individuals at the borders between the different regions. We still have to explore what causes this accumulation in 3D and how to circumvent it.

VI. Conclusion

Due to their inherent parallel search for multiple solutions and the consequential large computation times, parallelizing multi-objective evolutionary algorithms is an important issue. In this paper, we have argued that in addition to parallelizing single-objective evolutionary algorithms, in the multi-objective case there is the opportunity to divide up the search among the different processors, and have them search for different parts of the Pareto-optimal front.

We have then proposed the cone-separated NSGA-II, which explicitly divides up the search region by introducing additional constraints for each processor. First results on problems with two objectives were rather promising. We have shown that the approach is more efficient that the standard island model, and that it works independent of the shape of the Pareto-optimal front (even when it is concave and non-continuous). On problems with three objectives, the distribution of individuals was not as good, future work is necessary to address this problem. Also, we are currently looking into improved adaptation schemes (rather than just normalizing the objective space) and the effect of overlapping search regions. Furthermore, the ideas presented in this paper could also be used to bias the

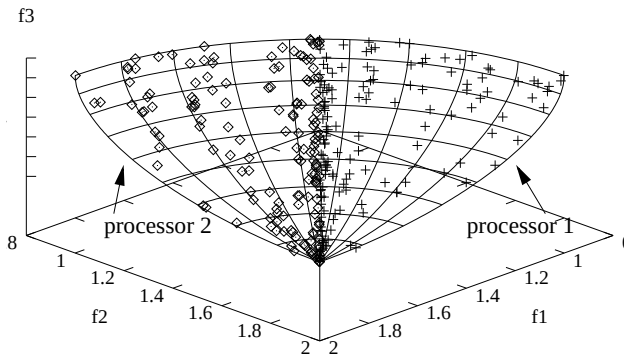

Fig. 11. Non-dominated front obtained with two processors for DTLZ2 problem.

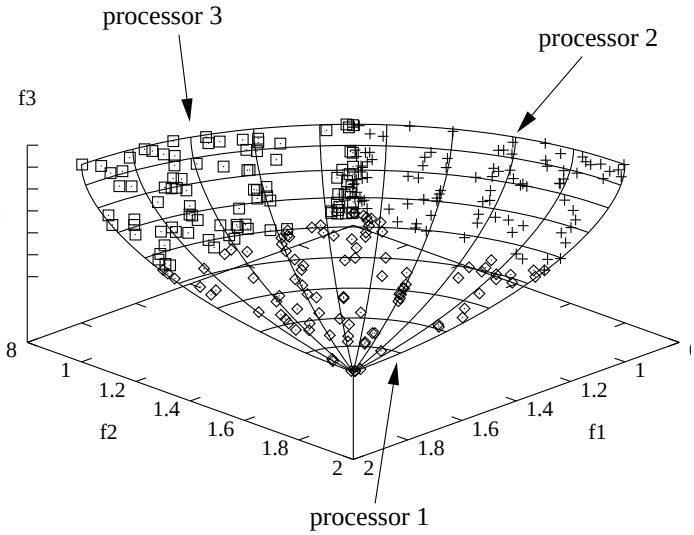

Fig. 12. Non-dominated front obtained with three processors for DTLZ2 problem.

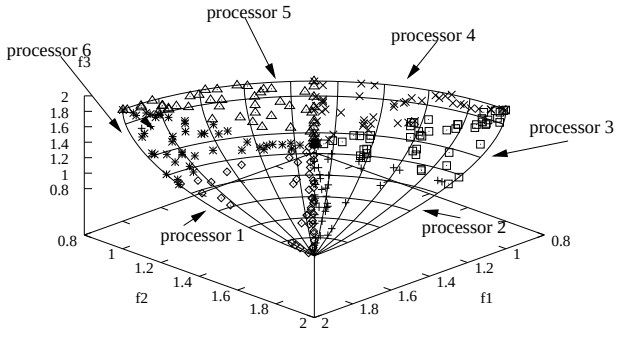

Fig. 13. Non-dominated front obtained with six processors for DTLZ2 problem.

search towards regions defined as interesting by a user (cf. [2]).

ACKNOWLEDGEMENTS:

K. Deb acknowledges the support through Bessel Research Award from Alexander von Humboldt Foundation, Germany during the course of this study.

REFERENCES

[1] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–461, 2002.

[2] J. Branke and K. Deb. Integrating user preferences into evolutionary multi-objective optimization. In *Knowledge Incorporation in Evolutionary Computation*. Springer, to appear.

[3] J. Branke, T. Kaußler, and H. Schmeck. Guidance in evolutionary multi-objective optimization. *Advances in Engineering Software*, 32(6):499–508, 2001.

[4] E. Cantu-Paz. *Effi cient and Accurate Parallel Genetic Algorithms*. Kluwer, 2000.

[5] F. de Toro, J. Ortega, J. Fernandez, and A. Diaz. Psfga: A parallel genetic algorithm for multiobjective optimization. In F. Vajda and N. Podhorszki, editors, *Euromicro Workshop on Parallel, Distributed and Network-Based Processing*, pages 384–391. IEEE, 2002.

[6] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(2):115–148, 1995.

[7] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature*, pages 849–858. Springer, 2000.

[8] K. Deb, L. Thiele, M. Laumanns, and E.Zitzler. Scalable test problems for evolutionary multi-objective optimization. Technical Report 2001001, Kanpur Genetic Algorithms Laboratory, Dept. of Mech. Engineering IIT Kanpur, India, 2001.

[9] 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*, volume 2632 of *LNCS*, pages 534–549. Springer, 2003.

[10] H. Horii, M. Miki, T. Koizumi, and N. Tsujiuchi. Asynchronous migration of island parallel ga for multi-objective optimization problem. In L. Wang, K. C. Tan, T. Furuhashi, J.-H. Kim, and X. Yao, editors, *Asia-Pacifi c Conference on Simulated Evolution and Learning*, pages 86–90. Nanyang Technical University, Singapore, 2002.

[11] R. Mäkinen, P. Neittaanmäki, J. Periaux, M. Sefrioui, and J. Toivanen. Parallel genetic solution for multiobjective mdo. In A. Schiano, A. Ecer, J. Periaux, and N. Satofuka, editors, *Parallel CFD'96 Conference*, pages 352–359. Elsevier, 1996.

[12] http://www-unix.mcs.anl.gov/mpi/.

[13] D. Quagliarella and A. Vicini. Sub-population policies for a parallel multiobjective genetic algorithm with applications to wing design. In *IEEE International Conference On Systems, Man, And Cybernetics*, volume 4, pages 3142–3147. IEEE, 1998.

[14] 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*, pages 47–66. Wiley, 2001.

[15] M. Miki T. Hiroyasu and S. Watanabe. The new model of parallel genetic algorithm in multi-objective optimization problems–divided range multi-objective genetic algorithm. In *Congress on Evolutionary Computation*, volume 1, pages 333–340. IEEE, 2000.

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

[17] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms. *Evolutionary Computation*, 8(2):125–148, 2000.

[18] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *Parallel Problem Solving from Nature*, pages 292–301. Springer, 1998.