A Hybrid Parallel Multi-Objective Genetic Algorithm: HybJacIsCone Model

Mahendra Kumar Gourisaria
School of Computer Engineering, KIIT University
Bhubaneswar -751024, Odisha, India

B.S.P.Mishra
School of Computer Engineering, KIIT University
Bhubaneswar -751024, Odisha, India

Satchidananda Dehuri
Ajou University
San 5 woncheon-dong
Suwon, South Korea

ABSTRACT

In real world most of the optimization problems are multi-objective in nature. These problems take large amount of time to congregate to the true Pareto front. So the basic algorithm like non parallel NSGA II may not able to solve such problem in ε-tolerable amount of time. This paper proposes a new hybrid parallel multi-objective genetic algorithm and solve one of the real life problem i.e., 0/1 knapsack problem. The proposed model is designed by combining the characteristics of Island model, Jakobovic model and Cone Separation model. It is experimented over a multi-core system and gives promising result over all the existing basic models in terms of converging to the true Pareto front.

Keywords

Parallel Multi-Objective Genetic Algorithm, Trigger Model, NSGA-II, Cone Separation Model, Island Model, 0/1 Knapsack Problem, HybJacIsCone Model

1. INTRODUCTION

Most of the engineering optimization problems have more than one objective which is contradictory to each other and these objectives must be fulfilled at the same time. Hence it is very difficult to find a single solution of it. We can call these types of problems as multi-objective optimization problem (MOP) [1]. Multi-objective problem gives a trade off solutions known as Pareto optimal solutions rather than giving only one optimal solution. From the trade off solutions, the user has to select a particular solution according to his own choice. In wider sense these solutions are optimal as no other better solution exists in the search space while considering all objective at a time. The Pareto optimal solution gives rise a Pareto Front having 'n' dimensional objective space where n represents the number of objectives in the problem. Genetic Algorithms (GAs) have the capability of exploring multiple Pareto optimal solutions in unit run, so it is widely used in this area.

In multi-objective genetic algorithm (MOGA), large number of solutions is required to be evaluated before finding out the promising result. Therefore it takes huge amount of time to converge in giving the solutions. So parallelization of MOGA can overcome such type of problem [11].

In parallelization of multi-objective genetic algorithms (MOGAs), multiple search space are explored by multiple processors to get different solutions. To parallelize genetic algorithm (GA), different models are proposed by different scientist like master slave model, Island model and cone-separation models [3, 4, 9].

In our paper we are proposing a hybrid model which can be operated in parallel environment by implementing multi-objective genetic algorithm (MOGA) for solving real world problems like 0/1 knapsack problem by considering convergence, divergence quality and time as the basic parameter.

The rest of the paper is organized as follows: Section 2 deals with basics of parallel computing. Section 3 deals with different parallel architecture and parallel multi-objective genetic algorithm (PMOGA) models. Section 4 describes the standard 0/1 knapsack problem. In section 5 we have presented the proposed hybrid model. The experimental analysis and conclusion is defined in Section 6 and 7 respectively.

2. BASICS OF PARALLELPROCESSING

Parallel computing solves problems using multiple computers, or computer with multiple internal processors at a greater computational speed than is possible with a uni-processor computer. Usually in case of a conventional computer, there is a single processor who performs the actions mentioned in a program. However, such a computer has limitation to tackle larger problems, i.e., problems with more computational steps or larger memory requirements. There are many ways of increasing the computational speed. One approach is that we can use multiple processors within a single computer (multiprocessor) or alternatively multiple computers, operating simultaneously on a single problem. In any one of the case, the whole problem is divided into different parts, and taken care by separate processor in parallel. We can write the programs/algorithms in the same way and this technique is known as parallel programming/algorithm. A parallel computer can contain multiple processors or more than one computer interconnected through an inter-connection networks. This type of system should also have more amount of total memory in comparison with single computer system to deal with the problem of larger memory requirement. But by this way, performance of the system should be increased. The scheme is that k processors/computers can offer up to k times the speed of computation of a single processor/computer, irrespective of the recent processor or computer speed, with the expectation that the problem would be finished in $(\frac{1}{k})^{th}$ of the time. Normally we cannot divide problems exactly into autonomous parts. Communication is also essential between the parts for synchronization of computations and data transfer. Though, we can achieve some improvement according to the problem and degree of concurrency inherent in the problem, by writing suitable parallel program. Today's parallel computer are much faster because of the frequent improvements in the processors

execution speed but still, present days computer cannot give any guarantee that they can solve any problem in a fair time period. The cost benefits of parallelism coupled with the performance requirements of applications areas such as numerical simulation, stock index prediction, and grand challenge problem like global weather forecasting, data mining present compelling arguments in favor of parallel computing. In order to obtain a valid result, these problems often required huge quantities of repetitive calculations on large amounts of data. Deadline is also an important factor in these kinds of problems i.e. computations work must be completed within a fair amount of time. We can take an example of weather forecasting. It is useless if our system is taking more than one day to predict the local weather for the subsequent day in an accurate manner. For the problem that require huge amount of main memory, multiple computers/processors are highly advisable which can also give increased speed. Sometimes, this situation also arises that a single problem has to be evaluated more than one time with different-different input values irrespective of the fact that the same problem can be solved in a fair time period. In case of parallel computer, this situation is especially applicable because different processors/computers can execute several instances of the same program simultaneously without any alteration to the program. Notwithstanding the continuous improvement in the speed of single computer, it is wrote by Flynn and Rudd that "the continued drive for higher and higher-performance systems ... leads us to one simple conclusion: the future is parallel" [7]. Algorithm development is a critical component of problem solving using computers. A sequential algorithm is nothing but a chain of fundamental steps for solving a given problem using a serial computer. Similarly, solving a given problem using more than one processor is told by parallel algorithm. However, specifying a parallel algorithm involves more than just specifying the steps. At the very least, a parallel algorithm has the additional height of concurrency and it is the duty of the person who designs the algorithm to specify all the set of steps that can be executed simultaneously. This is essential for obtaining any performance benefit from the use of a parallel computer. In practice, a nontrivial parallel algorithm may include a few or all of the following:

- Identifying that section of the work that can be performed in parallel.
- Mapping those sections of work which are concurrent in nature onto various processes which runs in parallel.
- Distribution of the input data, output data and in-between data related with the program.
- Data those are shared by more than one processor should be access in a managing way.
- The processors should be synchronized at different stages of the parallel program execution typically, there are several choices for each of the above steps, but usually, relatively few combinations of choices lead to a parallel data associated with the program.
- Managing accesses to data shared by multiple processors.
- Synchronizing the processors at various stages of the parallel program execution.

Typically, there are several choices for each of the above steps, but usually, relatively few combinations of choices lead to a parallel algorithm.

2.1 Speedup

Speedup can be described as the ratio between time taken to complete the job in a single processor and time taken to complete the job in a parallel processing environment. The speedup (s_p) is a measure of relative performance, which is thus defined as

$$S_p = \frac{t_s}{t_p}$$

We use t_s as the execution time of sequential algorithm running on a uni-processor and t_p is the execution time of the parallel algorithm for solving the equivalent problem with p multiprocessor.

2.2 Communication cost

We can define the communication cost as the total time taken for communication between two processor and the computation time, this can be defined as :

$$t_p = t_{comm} + t_{comp}$$

where t_{comm} is the communication time and t_{comp} is the computation time.

2.3 Efficiency

The cost of an ideal parallel system containing 'p' processing elements can deliver a speedup equivalent to 'p'. In practice, maximum speedup p is not achieved due to the process overhead. Efficiency can be defined as a measurement of the fraction of time for which a processing element is usefully employed. It is calculated as the ratio of speedup (S_p) to the number of processing elements (p) and denoted efficiency (E) by

$$E = S_p / p$$

3. RELATED WORKS

3.1 Parallel architecture

There are several schemes to categorize parallel computers [10, 8] have been suggested until now but not any of them can be treated as standard in the focused literature. These schemes differ on the basis of the characteristics of the parallel systems that are taken into consideration, namely: how the address space is organized, the interconnection network, or the granularity of the processors.

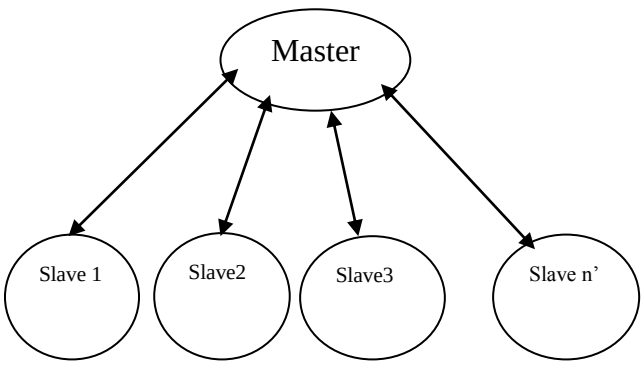

Fig 1: Schematic of master slave parallel GAs

The four classifications defined by Flynn are based upon the number of concurrent instruction (or control) and data streams available:

Single Instruction Stream and Single Data stream (SISD): A sequential computer comes under this category. These computers use no parallelism both in instruction and data stream.

Single Instruction and Multiple Data stream (SIMD): The computer which comes under this category uses single instruction stream and multiple data stream.

Multiple Instruction and Single Data stream (MISD): Multiple instructions work on a single data stream.

Multiple Instructions and Multiple Data stream (MIMD): Under this sachem, several independent processors concurrently execute on different data.

3.2 Parallelization models of MOEAs

The proposed parallelization schemes for MOEAs are resultant from the famous models designed for single-objective optimization [11]: the master-slave model, the diffusion model, the island model, and the hybrid model.

3.2.1 Master-Slave model

Under this model, we can parallelize MOEA in a very simple way and this model is also very popular. In this model, the job of the master processor is to execute the MOEA, and all the slave processors are busy in evaluating the objective function. The slave processors return the values of the objective function to the master after completion of their evaluation and remain inactive until the next generation. The master processors do the job of selection, crossover and mutation and also execute some tasks like Pareto ranking, and archiving. This model is shown in Figure 1. A master-slave parallel (pMOEA) explores the search exactly as a serial MOEA does. Thus, it locates the similar result found by its serial counterpart. Though, there is a substantial reduction in the execution timed.

3.2.2 Diffusion model

This model distributes the population between the neighborhood deme. For every grid point (as shown in Figure 2) there is one individual. This model is also called as fine grained model because there is one processor for every individual. Each deme has a processor and work on individual population. The selection and mating is restricted to a small area near by all individual. In this model the good character are spread or diffused all over the whole population because the neighborhoods are overlapped (as represented by the dotted lines in Figure 2). After standard interval the best populations are migrated between the demes. In this case the communication cost is very high. Due to this

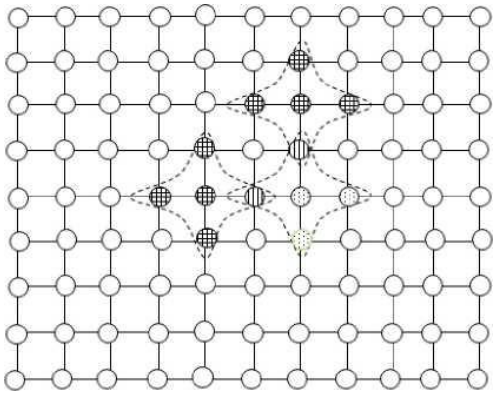

Fig 2: Diffusion model

diffusion model is suitable for Multiple Instruction Multiple Data computer.

3.2.3 Island model

In this model, using a separate sub-population every processor runs an in-dependent GA. The processors co-operate by exchanging the migrants (good individuals) on a regular basis. As the inter process communication is very limited, this model is mainly appropriate for computer clusters or grid computing system. Here in this model, we divide the population into a number of tiny sub-populations, which can be called as islands or demes which develop autonomously of each other. Every island run a serial MOEA for more than one generation called an epoch. After completion of every epoch, individuals migrate between nearby islands. Neighbors can be defined by the migration topology, which determines the path of migration along which individuals can shift to other islands. This model normally adopts a ring topology but other different topologies are also possible. Figure 3, shows a typical island model with a ring topology. As island pMOEA are generally implemented on distributed memory MIMD computers, that's why they are also known as distributed pMOEA.

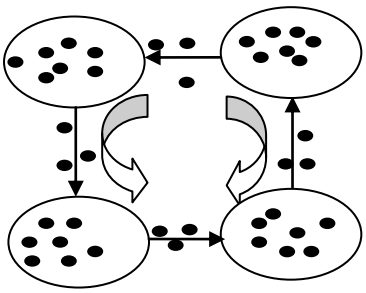

Fig 3: Island model

This model is extremely admired between researchers, but a lot of parameters and design decisions are required. The major issues in model are the migration topology, the frequency of migration, how many individuals to migrate, and the choice about the individuals who will migrate and those which will be changed by the immigrants.

3.2.4 Hybrid model

In this model, several basic models are combined together to give rise a hybrid model. Few of the hybrid models are shown in Figure 4

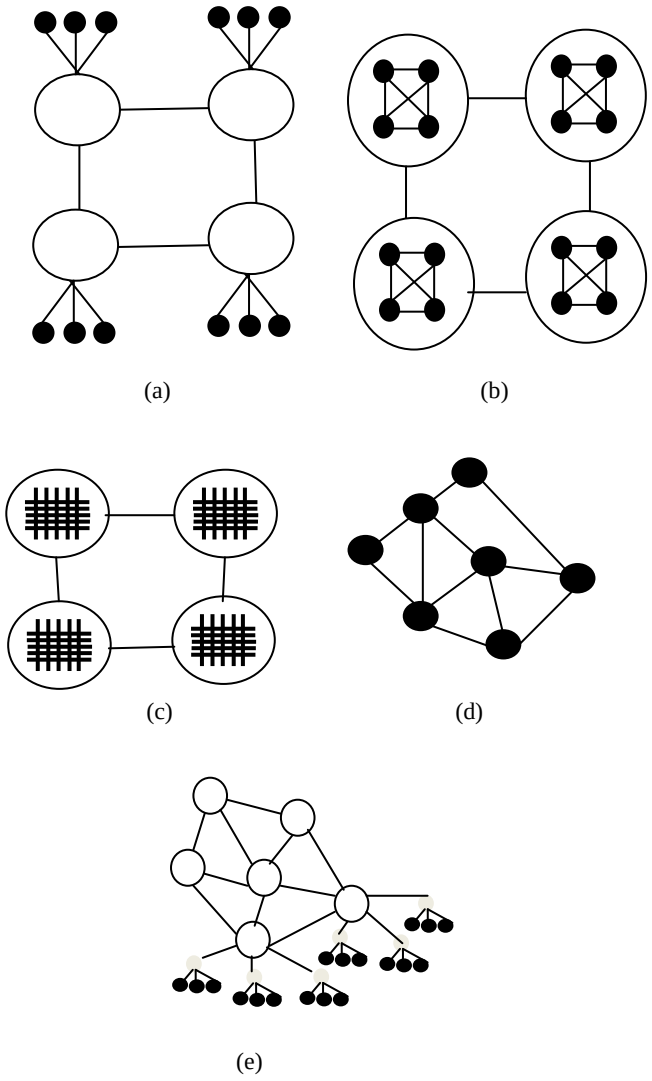

Fig 4: Different Hybrid models of PGA (a) coarse grain and global parallelization, (b) coarse grain and coarse grain, (c) coarse grain and fine grain, (d) peer-to-peer, (e) coarse grain, fine grain and global parallelization.

3.2.5 Cone-Separated NSGA-II

Branke et al. [3] implemented the concept of "divide-and-conquer" method to make all processors more efficient. Deb et al. [6] used the above technique by considering the guided dominance principle [2]. The above technique gave rise excellent result in the aspect of concurrency to the true Pareto front without guidance scheme. The major limitation of the above approach is to define suitable search direction before the shape of the Pareto front is known. In the above approach, the search spaces are divided in to several regions which are explored by multiple processors. The actual shape of the original Pareto front is unknown at the beginning of the optimization. So the portioning of the search space has to be done in a regular interval by normalizing the fitness value. After the normalization the partition of the cone is started from the reference point (1, 1). Each processor is assigned to the respective search space to explore the solution. Figure 5 illustrates the concept. In this, the border of the specified region are treated as constraints and taken care by using constraint dominance principle [5].

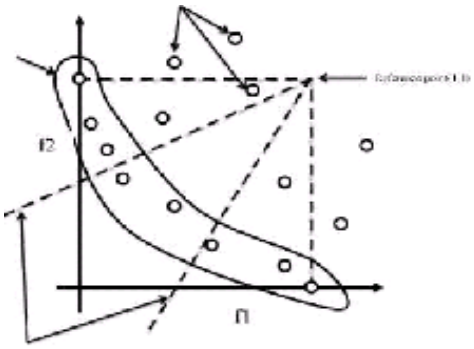

Fig 5: Example for the portioning of the (normalized) search space

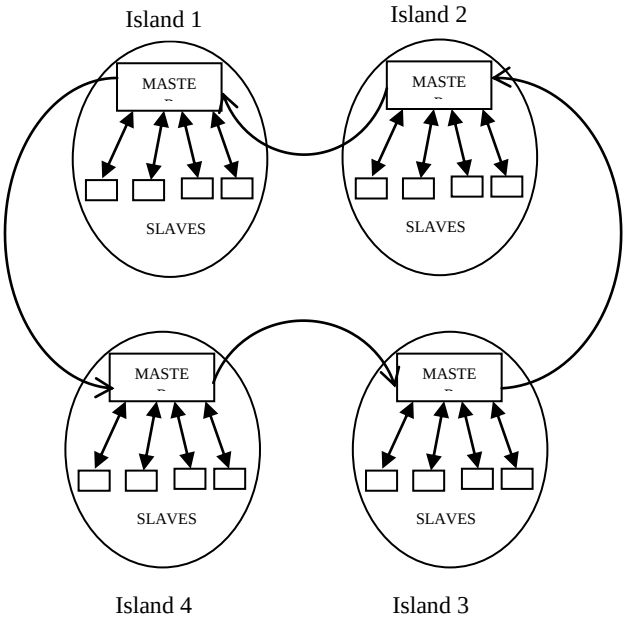

Fig 6: A HybJacisCone model

4. 0/1 KNAPSACK PROBLEM

One of the major NP hard problem is the 0/1 knapsack problem which is a maximization problem. The basic idea is finding out a set of items by considering the weight and profit associated with them, while the upper bound of the knapsack is the capacity of the knapsack. The main job is to find a subset of items which maximizes the total of profits in the subset, yet all the selected items fit into the knapsack, i.e. the total weight does not exceed the given capacity [12].

Let us assume that there are b number of item and d number of knapsack.

A single objective knapsack problem can be extended to a multi-objective problem by using two number of knapsack.

Formally, the multi-objective 0/1 knapsack problem is defined through Equation 1 and Equation 2.

Given a set of *b* items and a set of *d knapsacks*, with

$$p_{i,j} = profit\ of\ item\ j\ acording\ to\ knapsack\ i.$$
$$w_{i,j} = weight\ of\ item\ j\ according\ to\ knapsack\ i,$$
$$c_i = capacity\ of\ knapsack\ i.$$

Find a vector $x = (x_1, x_2 \ldots, x_b) \epsilon \{0,1\}^b$ such that,
$\forall\, i\,\{1, 2, ..., d\}: \sum_{j=1}^{b} w_{i,j} \cdot x_j \leq c_i$ (1)

and for which
$f(x) = (f_1(x), f_2(x), \ldots., f_d(x))$ is maximum, where
$f_i(x) = \sum_{j=1}^{b} p_{i,j}.x_j$ (2)
and $x_j = 1$ iff item j is selected.

5. PROPOSED MODEL

We can quickly converge to a set of good solutions in master-slave and island model when we apply parallel MOGA on it. but we can get better output if we combine both of them where each one is parallel in themselves. In our propose model, we have used island model at the upper level with Jakobovic master slave model at the lower level. In the lower level the Cone separation model is incorporated with the Jakobovic model. Figure 6 presents the pictographic representation of the algorithm.

5.1 HybJacIsCone Model

HybJacIsCone is a hybridization of Jakobovic, Island, and Cone separation model. In this, Jakobovic model takes care of population of each deme and all the demes communicate with each other at a standard interval by using ring topology. HybJacIsCone is hierarchal in nature with two levels, lower and higher level.

Lower level is hybridization of Jakobovic model and Cone separation model. Lower level is an independent island with more than one processor. Initially a Master is selected for each island and the remaining processors become Slaves for the respective islands. Each processor initialize different population and then Cone separation model is executed independently in every island as described in Algorithm 1. While executing for every *mig_within'* all slaves transfer its best *n'* individuals to respective master. Higher level is Island model. The master sort m' best individuals from the received population from the Slaves. The best m' individuals are propagate to the master of the right neighboring Island as shown in Algorithm 2. The receiving Master distributes the individuals equally to its Slaves.

6. EXPERIMENTAL STUDY

Experimental setup is presented in this section and this section also discusses the observed results of each experiment in detail.

Algorithm 1 Lower level algorithm for each island
Choose any one processer as a MASTER and others as SLAVE.
Sub-populations initializations
Fitness value normalization
Defining the region constrains
Non-dominated sorting
While (stopping-condition does not satisfy) do
if (Number of generation is divisible by *mig_within'*) then
  if SLAVE then
    Transfer the best n' individuals to the MASTER
  end if
  if (MASTER) then
    Receive the n' individuals from each SLAVE
    Do as in Algorithm 2
  end if
 end if
Generate Offspring
if (migration) then
Fitness value normalization
Defining the region constraints
Migrate individuals which contravene constraints
 end if
  Non-dominated sorting
  Pruning of population size
 end while

Algorithm 2 Higher Level algorithm for an island
Find m' number of best individuals
send m' individuals to the MASTER of the right neighboring island
Receive m' individuals from the MASTER of the left neighbor
Equally distribute the m' individuals among themselves i.e., MASTER and SLAVES

6.1 Experimental Setup

We have implemented the algorithm in C language on a multi-core system core i7 with 8 cores under Linux operating system. Each core is of 1.6 GHz. The RAM is 4GB. We have used MPICH (Message Passing Interface) library for communication between the processors. Table 1 shows the parameters of the parallel MOGA in which the first row explains the population size per deme. Crossover and mutation probability are shown in second and third row respectively. Row four shows the migration rate for the Cone separation model. The other parameters for HybJacIsCone model is in row fifth and sixth.

6.2 Experimental Results

We have conducted the experiment with multi-objective 0/1 knapsack problem with 8 processors, 2 knapsacks, and 200 data items. The problem is solved by proposed hybrid model. We

TABLE 1. Parameter set

Population size / Deme	200
Crossover probability	0.8
Mutation probability	0.016/bit
Migration rate for Cone Separation	After every 10 generation
mig-within!	10 generations
m′, n′	20 individuals
No of Processors	8
Termination condition	Average knapsack profit greater than 10000 or the migration of non-dominated solution remain dormant for 20 generations

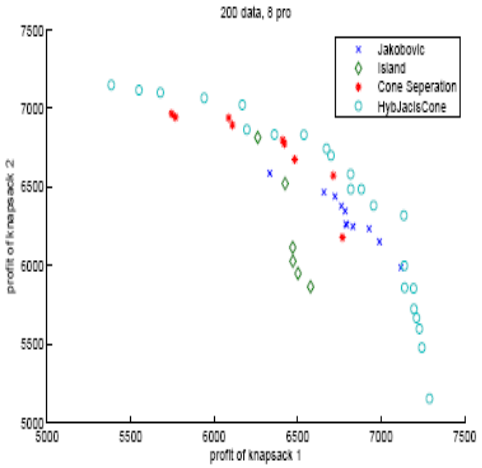

Figure 7: Comparison of divergence and convergence with different PMOGA models with 200 data items and 8 processors.

have compared the result with the jakobovic, Island and Cone separation model. Figure 7 explains the divergence result with different models. From the Figure 7 it can be seen that HybJacIsCone model diverges better than any of the independent models.

7. CONCLUSION

Multi-objective genetic algorithm (MOGA) parallelization is an important issue due to its large computation time with several solutions. Our paper presents a hybrid model in the direction of parallelizing multi-objective genetic algorithm (MOGA) and it is verified over 0/1 knapsack problem. In addition to this, the result is compared with the basic model. Once more the efficiency of the proposed model is verify over the divergence parameter and found out that it gives better result in comparison with existing models by varying the number of processors.

8. REFERENCES

[1] T. Al-Somani and K. Qureshi. *Reliability Optimization Using Genetics Algorithms*. Msc thesis, Saudi Arabia, King Abdul Aziz University.2000

[2] J. Branke, T. Kaubler, and H. Schmeck. Guidance in evolutionary multiobjective optimization., *Advances in Engineering Software, volume 32(6), pages 499-508,2001*

[3] J. Branke, H. Schmeck, K. Deb, and R. S. Maheshwar. Parallelizing multi-objective evolutionary algorithms: Cone separation. In *Congress on Evolutionary Computation (CEC 2004),* pages 1952-1957, Portland, Oregon, USA, 2004. IEEE Press

[4] E. Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, volume* 10(2), pages 141-171, 1998

[5]] K. Deb, S. Agarwal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature,* pages 849-858. Springer-Verlag, 2000.

[6] K. Deb, P. Zope, and A. Jain. Distributed computing of pareto-optimal solutions with evolutionary algorithm. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization, LNCS,* volume 2632, pages 534-549. Springer, 2003

[7] M. J. Flynn and Kevin W. Rudd. Parallel architectures. *ACM Comput. Survey ,volume* 28, pages 67-70, March 1996.

[8] E. E. Johnson. Completing an mimd multiprocessor taxonomy. *SIGARCH Computer Architecture News, volume* 16(3), pages 47-44, 1988.

[9]] F. Streichert, H. Ulmer, and A. Zell. Parallelization of multi-objective evolutionary algorithms using clustering algorithms. *Evolutionary Multi-Criterion Optimization, volume* 3410, pages 92-107, 2005

[10]] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principals and Paradigms.* Prentice Hall, Upper Saddle River, 2002

[11] D. A. van Veldhuizen. Considerations in engineering parallel multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation, volume* 7(2), pages 144-173, 2003

[12] E. Zitzler, K. Deb, and L. Thiele. Comparison of multi-objective evolutionary algorithms: Empirical results. Technical report, INSTITUTION Swiss Federal Institute of Technology (ETH), Zurich, 1999