# Parallel Multi-Objective Genetic Algorithm
## GPU Accelerated Asynchronously Distributed NSGA II

Oliver Rice, Robert E. Smith, and Rickard Nyman

University College London (UCL)
London WC1E 6BT, UK
Oliver@oliverrice.com,
Robert.Elliott.Smith@gmail.com,
RickardNyman@gmail.com

**Abstract.** Multi-objective optimization problems consist of numerous, often conflicting, criteria for which any solution existing on the Pareto front of criterion trade-offs is considered optimal. In this paper we present a general-purpose algorithm designed for solving multi-objective problems (MOPS) on graphics processing units (GPUs). Specifically, a purely asynchronous multi-populous genetic algorithm is introduced. While this algorithm is designed to maximally utilize consumer grade nVidia GPUs, it is feasible to implement on any parallel hardware. The GPU's massively parallel architecture and low latency memory result in +125 times speed-up for proposed parametrization relative to single threaded CPU implementations. The algorithm, NSGA-AD, consistently solves for solution sets of better or equivalent quality to state-of-the-art methods.

**Keywords:** genetic algorithm, multi-objective optimization, GPU acceleration, parallel computing.

## 1 Introduction

Graphics Processing Units (GPUs), initially designed for graphics rasterisation, are increasingly receiving attention in the scientific computing community. This interest spurs from the massive raw compute capabilities associated with GPU's highly parallel architecture. The subsequent advent of general purpose GPU computing languages such as Compute Unified Device Architecture (CUDA) [13] and Open Compute Language (OpenCL) [12] have made GPU acceleration of computationally expensive algorithms an attractive prospect. Both CUDA and OpenCL are extensions of the C programming language.

The multiple layers of parallelism within genetic algorithms have made them prime candidates for GPU acceleration in the past. Beyond simply parallelizing the fitness function for each candidate solution, further efforts also parallelize selection, recombination and mutation operators [4]. Several categories of the resulting algorithms include island based [10], distributed [3], and cellular [2]. These algorithms have been designed and tested for various parallel hardware environments including CPU clusters, FPGAs [11] and GPU clusters [9] with

A.-H. Dediu et al. (Eds.): TPNC 2013, LNCS 8273, pp. 217–227, 2013.

applications ranging from shop scheduling [1], biology [15], chemistry [18] and finance [17].

Despite proposals for parallelizing single-run multi-objective genetic algorithms [7], most multi-objective implementations either involve highly domain specific approaches [16,6] or restricted selection models. In these instances MOPS are decomposed to multiple scalar optimization problems (SOPs) and optimized independently [8]. The goal of this paper is to parallelize computation of the entire Pareto optimal frontier. To this end we propose a cooperative approach in which multiple populations are independently processed to find predefined subsections of the Pareto front. This is achieved through use of an innovative decomposition of fitness functions allowing implementation of an NSGA II variant for multiple asynchronously computed sub-populations. Each of these sub-populations is a speciating island which maps to a portion of the Pareto frontier. We call the algorithm nondominated sorting genetic algorithm asynchronously distributed (NSGA-AD).

In section 2 the GPU memory model is discussed. Following, in section 3 an outline of NSGA II is presented. Section 4 introduces the trade-offs associated with different GA memory mappings to the GPU. Section 5 combines the previous information in development of our proposed algorithm which is tested according to quality and speed in sections 6 and 7 respectively. We end with conclusions in section 8.

## 2 GPU Memory Model

General purpose GPU programming remains in its infancy. Given this is the target hardware for the proposed algorithm we first discuss the internal memory speeds and limitations of the GPU.

From the CPU, data may be pushed to global, constant and texture memory banks. The speed of this transaction can be limited by either the graphics card's memory bus or the host system's motherboard PCI-E port. The maximum speeds of PCI-E versions 1, 2, and 3, are 4, 8, and 16 GB/s respectively.

The paradigm for parallel GPU computing segments operations first into blocks, and then to threads. As of CUDA 5.0, each block may contain up to 1024 threads which have access to 6 memory types. Global memory or DRAM is the largest bank with 2GB on the consumer grade nVidia GeForce GTX 660 Ti used to evaluate GPU applications in this paper. Global memory is located off chip and resultantly has access latency on the order of 100 times less than on chip shared and register memory. Shared memory is accessible from any thread in the same block but is limited to 48KB. Registers, in contrast, have thread scope and are limited to 63 32-bit registers per thread. For our purposes, local memory can be thought of as low-speed spillover if register memory is filled. Finally, constant and texture memory can be ignored for the proposed implementation purposes.

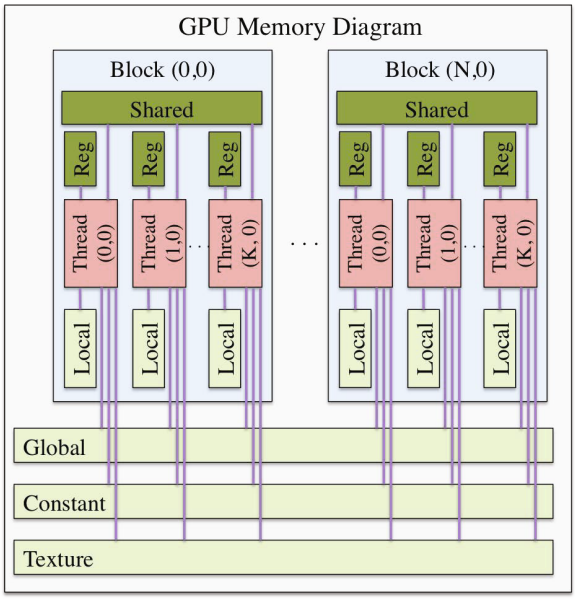



**Fig. 1.** GPU Memory Diagram

## 3 NSGA II

For multi-objective GAs (MOGA) nondominated sorting genetic algorithm II (NSGA II) is widely considered state-of-the-art [19]. Several key advantages over other popular MOGAs include, reduced computational complexity, elitism, and parameterless fitness sharing operator [5]. The algorithm can be outlined as follows.

1. Random population $P_0$ is created of size N
2. Evaluate $P_i$ for all fitness criteria
3. Sort $P_i$ by nondomination & crowd distance
4. Selection, Recombination, Mutation
5. Resulting population is $Q_i$ of size N
6. $R_i = append(P_i, Q_i)$
7. Sort $R_i$ by nondomination & partial rank
8. $P_{i+1}$ = first N in $R_i$
9. Repeat steps 2-8 until exit criteria

where N is the population size.

Nondomination level is assigned such that no individuals on the same level strictly dominate any other individuals along all fitness axes. In other words, the first level consists of the population's best estimate of the Pareto set, and each subsequent level is the Pareto set of the population excluding all individuals contained by lower levels.

To maximize the distance among found points on the current population Pareto front, each level is internally sorted according to the crowding distance between its nearest neighbors. In this context, crowding distance is computed by sorting the population first on level, and then on each of the fitness function values in ascending order. When the currently sorted fitness function is c, and level is l the crowding distance is found from equation 1.

$$\text{Dist (i)} = \frac{1}{2} \sum_{c=1}^{2} \sum_{i=\min(l)\ +1}^{i=\max(l)\ -1} \frac{P_{l,i+1}.c - \ P_{l,i-1}.c}{\max\left(P_l.c\right)\ - \min(P_l.c)} \tag{1}$$

where $P_{(l,i)}.c$ is the value of fitness function c for the $i^{th}$ individual in level l and $max(P_l.c)$ is the maximum fitness value for the $l^{th}$ level of population P.

The individuals with the minimum and maximum fitness values for each level are given an infinite distance. In effect this makes the extremes of each level most likely to be selected for recombination. Even considering NSGA II's diminished computational complexity relative to NSGA it remains impractical for large population sizes. As originally proposed NSGA II is also memory intensive. Namely, in order to reduce the time/computational complexity, storage requirements become $O(N^2)$ which can further limit population size in memory restricted environments.

## 4 Memory Footprint

In several key areas GPUs are ideally suited to solve genetic algorithms. One primary advantage is the capability to produce random numbers in register memory. The CURAND library does exactly this by allowing each thread to seed its own random number generator, produce, and finally consume random numbers without use of slow-access global memory. This fact, combined with GAs' heavy reliance on random numbers in the selection, recombination, and mutation phases can lead to significant performance gains.

Beyond random number generation, the primary consideration when discussing island based, or decomposed GAs, is memory management. The ideal case is to house all populations in unique blocks of high speed shared memory. This can prove problematic due to the restrictive storage capacity of 48KB. As seen in figure 2, 48KB is a severe limitation for most genome types. Note that computation of the maximum genome length under the heading 'Max G NoVar' contains raw population storage only, while 'Max G Var' incorporates 16 bytes per genome and 32 bytes per population of working variables.

Instances of research which address the memory issue where complex genome types and large population sizes are required most commonly utilize global memory for genome storage. Doing so all but eliminates memory considerations, but access latency suffers dramatically.

In scenarios where fitness functions are sufficiently computationally expensive it may be possible to partially hide memory access lag [14]. When in combination

| Population Size | | | | Max G | |
|---|---|---|---|---|---|
| bitset | bool | short | int/float | NoVar | Var |
| 128 | 16 | 8 | 4 | 1500 | 1483 |
| 256 | 32 | 16 | 8 | 750 | 733 |
| 512 | 64 | 32 | 16 | 375 | 358 |
| 1024 | 128 | 64 | 32 | 187 | 171 |
| 2048 | 256 | 128 | 64 | 93 | 77 |
| 4096 | 512 | 256 | 128 | 46 | 30 |
| 8192 | 1024 | 512 | 256 | 23 | 7 |
| 16384 | 2048 | 1024 | 512 | 11 | 0 |
| 32768 | 4096 | 2048 | 1024 | 5 | 0 |

**Fig. 2.** Genome Memory Limitations

with parallel sorting and nondomination functions, the GPU can be fully utilized for the majority of genetic algorithm phases.

We propose two technical implementation differences to the usual shared-memory, island-style model which partially alleviate the restricted genome size. First, a bitset is used to enable binary values to be stored at single bit resolution. A side effect when genes are binary is a reduction in memory requests as each 32-bit variable contains 32 genes. For non-binary genome types the bitset can be manipulated to ensure dense packing of any data type to a specified resolution. The second difference is the dynamic 'as-needed' use of global memory. Given the knowledge that lowest level and least crowded solutions are most likely to be selected for recombination, these solutions require the most memory accesses. To reduce global memory accesses, the frequently retrieved solutions are housed in shared memory to the maximum extent possible. The remaining less frequently accessed genomes are allocated in global memory.

## 5 NSGA-AD

As previously mentioned, the goal of this NSGA II adaptation is to produce an island-model in which each island or sub-population maps a section of the Pareto optimal frontier. Scalar approaches to decomposition convert MOPs to multiple SOPs by providing weight vectors to each sub-population. The sub-populations' fitness figures are then evaluated by linearly weighting all fitness values for each individual via the corresponding element of the relevant weight vector. A simple example is shown in figure 3. Note that the appeal this structure is its ease of parallelization on a population granularity. Parallelization of NSGA II is not as straightforward. Applying multiple parallel populations with NSGA II would cause each population map the entire Pareto frontier as in figure 4.

We suggest decomposing the Pareto front similarly to the scalar decomposition approach. Instead of passing a single weight vector, the search space is linearly divided into equal portions by a weight range.

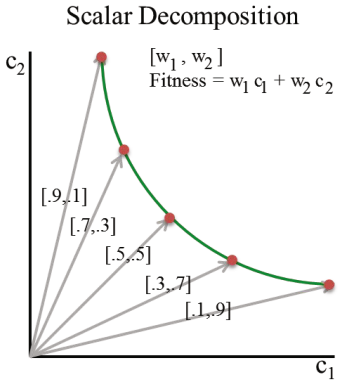



**Fig. 3.** Scalar Decomposition of Multi-Objective Search Space

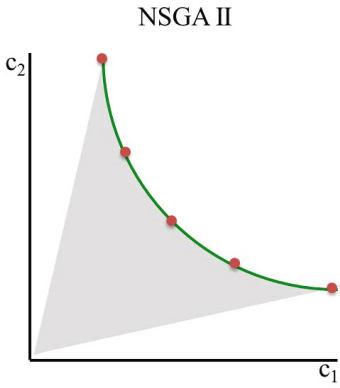



**Fig. 4.** NSGA II in Multi-Objective Search Space

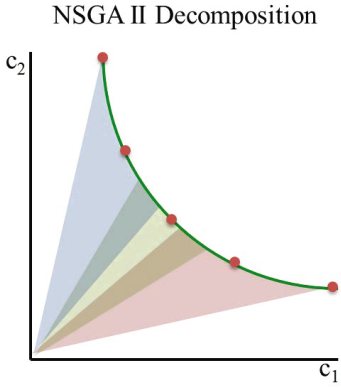



**Fig. 5.** Proposed Decomposition of Multi-Objective Search Space

To enforce each sub-population tracts to its desired section of the search space, a constraint parameter is introduced. This parameter is such that a solution's genome is within its constraint when its host sub-population's range of weight vectors yields the highest value of Fitness in the following.

$$Fitness = \sum_{i=1}^{C} (w_{d,i} * c_{d,i}) \tag{2}$$

where d is the sub-population index and i the index of each fitness criterion.

Each sub-population has a target area of the C dimensional search space which it is responsible for. This can be seen in figure 5. Once the standard NSGA II sort is complete, each solution's fitness values are entered into equation 3 with each population's weights. When the solution's host population weights yield the highest total fitness value relative to all other weights the solution is said to be within its target space. If any other set of weights yields a higher total value then the solution is outside its target space. Explicitly, the constraint parameter V for each individual in a sub-population is found by:

$$V = \begin{cases} 0 \sum_{i=1}^{C} (w_{d,i} * c_{d,i}) & > \sum_{i=1}^{C} (w_{j,i} * c_{j,i}) \\ 1 & otherwise \end{cases} \tag{3}$$

For all sub-populations j where $j \neq d$.

The constraint is introduced as selective pressure in each sub-population via an adjustment to the nondomination sort. Any individual which does not satisfy V==0 is decremented a single nondomination level after the initial sort. It would be feasible to treat constraints as additional fitness metrics when performing the NDS. However, this implementation requires a further, computationally expensive, sort.

## 6 Performance: Quality

The test problems selected to review performance statistics were drawn from Zitzlers proposed test problems [20]. From the six suggested problems we select three for use, ZDT1, ZDT2, and ZDT3.

These three examples were chosen as they each exhibit unique properties in Pareto frontier shape. ZDT1 (figure 6) has a convex and continuous Pareto front. ZDT2 (figure 7) is nonconvex and continuous while ZDT3 is convex and discontinuous (figure 8) [1].

With respect to the quality of produced solutions, we intend only to demonstrate the proposed algorithm yields equivalent results to NSGA II. For this purpose evaluation of convergence and diversity metrics are performed according to the methodology given in [5] to permit direct comparison of originally described performance.

[1] Pareto front applicable only between x = [0.00000,0.08300], [0.18222,0.25776], [0.40931,0.45388], [0.61839,0.65251], [0.82333,0.85183]

ZDT1

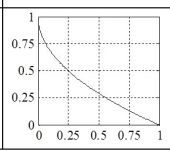
| Fitness Functions | Constraints | Plot |
|---|---|---|
| $f_1(x) = \quad x_1$<br>$f_2(x) = \quad g(x) * \left(1 - \sqrt{\frac{x}{g(x)}}\right)$<br>$g(x) = \quad 1 + \frac{9}{n-1} * \sum_{i=2}^{G} x_i$ | $0 \le x_i \le 1$<br>i= 1, 2, ..., G<br>G=30 |  |
| | Pareto Front | |
| | $x_2 = 1 - \sqrt{x_1}$ | |

**Fig. 6.** ZDT1 Test Function

ZDT2

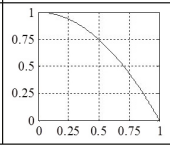
| Fitness Functions | Constraints | Plot |
|---|---|---|
| $f_1(x) = \quad x_1$<br>$f_2(x) = \quad g(x) * \left(1 - \left(\frac{x_1}{g(x)}\right)^2\right)$<br>$g(x) = \quad 1 + \frac{9}{n-1} * \sum_{i=2}^{G} x_i$ | $0 \le x_i \le 1$<br>i= 1, 2, ..., G<br>G=30 |  |
| | Pareto Front | |
| | $x_2 = 1 - {x_1}^2$ | |

**Fig. 7.** ZDT2 Test Function

ZDT3

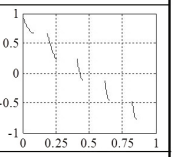
| Fitness Functions | Constraints | Plot |
|---|---|---|
| $f_1(x) = \quad x_1$<br>$f_2(x) = \quad g(x) * (1 - \sqrt{x_1/g(x)} - \sin(10\pi x_1))$<br>$g(x) = \quad 1 + \frac{9}{n-1} * \sum_{i=2}^{G} x_i$ | $0 \le x_i \le 1$<br>i= 1, 2, ..., G<br>G=30 |  |
| | Pareto Front | |
| | $x_2 = 1 - \sqrt{x_1} - x_1 \sin(10\pi x_1)$ | |

**Fig. 8.** ZDT3 Test Function

Convergence quality is determined by selecting 500 points on the known Pareto optimal frontier designated by set H. Minimum Euclidean distance from each individual in the solution set produced by the GA to its nearest counterpart contained in set H is computed. This metric is designated by $\Upsilon$.

The second metric of diversity measures the extent to which the solution set is distributed across the Pareto front. Both distribution within the extreme

points of the optimal set and Euclidean distance between the extreme values are considered in accordance with the diversity metric in equation 4.

$$\triangle = \frac{d_f + d_t + \sum_{i=1}^{U(0)-1} \left| d_i - \overline{d} \right|}{d_f + d_t + \overline{d_t}\,(U\,(0) - 1)} \tag{4}$$

where $d_f$ and $d_t$ represent Euclidean distances between maximally distant solutions in the Pareto set and solved for set respectively.
U(0) is the number of nondominated solutions
$\overline{d}$ is taken to be the average Euclidean distance between each nondominated solution and its nearest neighbor notated $d_i$ along the found frontier.

Mean and variance for both metrics were obtained from 10 independent runs with unique seeds to each threads' random number generator. Parametrization of run limits was set to 25,000 function evaluations per population for both NSGA II and NSGA-AD. Decomposition of the fitness functions in sub-populations of NSGA-AD was fixed such that each of 16 parallel populations received $(1/14)^{th}$ of the linearly divided fitness space. Note the marginal overlap resulting from this division.

It is necessary to incorporate multiple sub-populations when evaluating performance statistics because a single-island implementation of NSGA-AD is identical to the original NSGA II algorithm. The purpose of the quality test is to determine if dissection of the Pareto front negatively impacts convergence or diversity of solutions. Figure 9 shows the performance of the binary encoded NSGA II and binary encoded NSGA-AD algorithms with each set of test functions.

| **Algorithm** | | **ZDT1** | | **ZDT2** | | **ZDT3** | |
|---|---|---|---|---|---|---|---|
| **Metric** | | **ϒ** | **Δ** | **ϒ** | **Δ** | **ϒ** | **Δ** |
| **NSGA II** | **Mean** | 0.000894 | 0.463292 | 0.000824 | 0.435112 | 0.043411 | 0.575606 |
| **(Binary)** | **Var** | 0.000000 | 0.041622 | 0.000000 | 0.024607 | 0.000042 | 0.005078 |
| **NSGA-AD** | **Mean** | 0.000822 | 0.386640 | 0.000826 | 0.385270 | 0.006901 | 0.415940 |
| **(Binary)** | **Var** | 0.000000 | 0.011585 | 0.000000 | 0.009852 | 0.000000 | 0.018810 |

**Fig. 9.** Performance Metrics

For test problems ZDT1 and ZDT2 the convergence values are lower-bound-limited by incidental distances to the 500 uniformly selected points on the true Pareto front. The values are small enough such that the variance between runs rounds to zero. NSGA-AD demonstrates significantly improved performance relative to NSGA II on the discontinuous ZDT3 fitness function. This can most likely be attributed to the increased ability to speciate when mapping independent populations to subsets of the frontier. We leave exploration of any potential quality gains for a future study and focus primarily on computational speed advantages.

## 7 Performance: Speed

Speed performance of the GPU ported NSGA II variant is measured on a NVIDIA GeForce GTX 660 Ti GPU which contains 1344 computational cores clocked to 980 MHz. For comparison, the same algorithm is tested on an Intel Core i7 950 CPU clocked to 3.06 GHz.

| Pop | Parallel Sub-Populations | | | | |
|---|---|---|---|---|---|
| Size | 1 | 2 | 4 | 8 | 16 |
| 32 | 0.2 | 0.3 | 0.6 | 1.2 | 2.2 |
| 64 | 0.3 | 0.6 | 1.2 | 2.4 | 4.7 |
| 128 | 0.8 | 1.6 | 3.2 | 6.3 | 12.3 |
| 256 | 2.2 | 4.4 | 8.6 | 17.0 | 18.6 |
| 512 | 7.1 | 14.0 | 27.8 | 30.8 | 39.6 |
| 1024 | 27.8 | 31.0 | 31.4 | 31.9 | 32.4 |
| 2048 | 128.0 | 128.2 | 129.2 | 129.7 | 129.7 |

**Fig. 10.** Speedup Results: GPU vs CPU

The speed-ups shown in figure 10 represent the GPU implementation versus a single threaded CPU implementation. All GPU code was written to allow perfect scalability from serial execution to parallelism on the order of 1 thread per solution candidate. Coding for the possibility of serial execution allowed the C code utilized in the CPU implementation to overlap perfectly with its equivalent CUDA code for +95% of the program.

A maximum speed up of 129.7 times was obtained using 16 parallel sub-populations with each sub-population containing 2048 individuals.

## 8 Conclusions

Through use of a GPU we have demonstrated it is possible to attain positive speed-ups for multi-objective genetic algorithms with outputs equivalent to NSGA II. When using the proposed algorithm speed-ups were realized at all tested population sizes with +8 parallel sub-populations. The speed improvement in figure 10 can be seen to level off when the number of parallel sub-populations multiplied by the population size exceeds the number of cores within the GPU. This occurs when the utilization of GPU resources becomes bottlenecked under the described implementation. The proposed algorithm was found to perform at least as well as NSGA II in all test cases along selected quality metrics while generating a maximum of 129.7 times speed up.

## References


1. Akhshabi, M., Haddadnia, J., Akhshabi, M.: Solving flow shop scheduling problem using parallel genetic algorithm. Procedia Technology 1, 351–355 (2012)
2. Alba, E., Dorronsoro, B.: Computing nine new best-so-far solutions for Capacitated vrp with cellular Genetic Algorithm. Information Processing Letters 98, 225–230 (2006)
3. Alba, E., Troya, J.M.: Analyzing synchronous and asynchronous parallel distributed genetic algorithms. Future Generation Computer Systems 17, 451–465 (2001)
4. Davies, R., Clarke, T.: Parallel implementation of a genetic algorithm. Control Engineering 3, 11–19 (1995)
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6 (2002)
6. Duran, J.P., Kumar, S.A.: CUDA based multi objective parallel genetic algorithms: Adapting evolutionary algorithms for document searches (unpublished)
7. Durillo, J., Nebro, A., Luna, F., Alba, E.: A study of master-slave approaches to parallelize nsga-ii. In: IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, pp. 1–8 (2008)
8. Gustafson, S., Burke, E.K.: The speciating island model: An alternative parallel evolutionary algorithm. Journal of Parallel and Distributed Computing 66, 1025–1036 (2006)
9. Jaros, J.: Multi-gpu island-based genetic algorithm for solving the knapsack problem. World Congress on Computational Intelligence (June 2012)
10. Maeda, Y., Ishita, M., Li, Q.: Fuzzy adaptive search method for parallel genetic algorithm with island combination process. International Journal of Approximate Reasoning 41, 59–73 (2006)
11. Moreno-Armendariz, M.A., Cruz-Cortes, N., Duchanoy, C.A., Leon-Javier, A., Quintero, R.: Hardware implementation of the elitist compact Genetic Algorithm using Cellular Automata pseudo-random number generator. Computers and Electrical Engineering (2013)
12. nVidia: OpenCL Programming Guide for the CUDA Architecture (2009), `http://www.nvidia.com/content/cudazone/download/OpenCL/NVIDIA_OpenCL_ProgrammingGuide.pdf`
13. nVidia: CUDA C Programming Guide (2012), `http://docs.nvidia.com/cuda/cuda-c-programming-guide/`
14. Pospichal, P., Jaros, J.: Gpu-based acceleration of the genetic algorithm, gECCO Competition (2009)
15. Rausch, T., Thomas, A., Camp, N.J., Cannon-Albright, L.A., Facelli, J.C.: A parallel genetic algorithm to discover patterns in genetic markers that indicate predisposition to multifactorial disease. Computers in Biology and Medicine 28, 826–836 (2008)
16. Solar, M., Parada, V., Urrutia, R.: A parallel genetic algorithm to solve the set-covering problem. Computers & Operations Research 29, 1221–1235 (2002)
17. StraBburg, J., Gonzalez-Martel, C., Alexandrov, V.: Parallel genetic algorithms for stock market trading rules. Procedia Computer Science 9, 1306–1313 (2012)
18. Tantar, A., Melab, N., Talbi, E.G., Parent, B., Horvath, D.: A parallel hybrid genetic algorithm for protein structure prediction on the computational grid. Future Generation Computer Systems 23, 398–409 (2007)
19. Zhou, A., Qu, B.Y., Li, H., Zhao, S.Z., Suganthan, P.N., Zhang, Q.: Multiobjective evolutionary algorithms: A survey of the state of the art. Swarm and Evolutionary Computation 1, 32–49 (2011)
20. Zitzler, E., Deb, K., Thiele, L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Evolutionary Computation 8(2), 173–195 (2000)