GPU-Accelerated Differential Evolutionary Markov Chain Monte Carlo Method for Multi-Objective Optimization over Continuous Space

Weihang Zhu
Department of Industrial Engineering
Lamar University
211 Redbird Ln, P.O. Box 10032, Beaumont, TX 77710
1-409-880-8876
Weihang.Zhu@lamar.edu

Yaohang Li
Department of Computer Science
North Carolina A&T State University
1601 East Market St., Greensboro, NC 27411
1-336-334-7245
yaohang@ncat.edu

ABSTRACT

In this paper, the attractive features of evolutionary algorithm and Markov Chain Monte Carlo are combined into a new Differential Evolutionary Markov Chain Monte Carlo (DE-MCMC) method for multi-objective optimization problems with continuous variables. The DE-MCMC evolves a population of Markov chains through differential evolution (DE) toward a diversified set of solutions at the Pareto optimal front in the multi-objective function space. The computational results show the effectiveness of the DE-MCMC algorithm in a variety of standardized test functions as well as a protein loop structure sampling application. Moreover, the DE-MCMC algorithm can efficiently take advantage of the massive-parallel, many-core architecture, where its implementation on GPU can achieve speedup of 14~35.

Categories and Subject Descriptors

G.1.0 [**Mathematics of Computing**]: General – *parallel algorithms*. G.1.6 [**Mathematics of Computing**]: Optimization.

General Terms

Algorithms, Performance.

Keywords

Multi-objective Optimization, Markov Chain Monte Carlo, Graphics Processing Units, Differential Evolution

1. INTRODUCTION

Markov Chain Monte Carlo (MCMC) methods have been used as indispensable tools to produce random samples from complex and intractable distributions in a variety of problems. The fundamental algorithm was proposed by Metropolis et al. [1] and generalized by Hastings [2] by incorporating a transition kernel. The rationale of MCMC is to generate a Markov chain by generating dependent samples with target distribution as its equilibrium distribution. Recent advances of MCMC method include simulated annealing [3], simulated tempering [4], parallel tempering [5], generalized-ensemble Monte Carlo [6], multiple try Metropolis [7], hybrid Monte Carlo [8], configuration-biased Monte Carlo [9] and many others. In some situations, an optimization problem can also be formulated as a Monte Carlo sampling problem, where solutions located within the vicinity of the global minimum of the objective function have the highest probability to be reached during the sampling process. Therefore, MCMC is also popularly used as a stochastic optimization procedure in many scientific computing applications, including physics [10], chemistry [11], biology [12], and material science [13].

In most applications where MCMC is used for optimization, the goal is to find the global minima or maxima of a single objective function. In 2002, Vrugt et al. extended MCMC to multi-objective optimization by employing the concept of Pareto dominance, where the corresponding Multiobjective Shuffled Complex Evolution Metropolis (MOSCEM) algorithm [14] has demonstrated certain effectiveness and efficiency in multi-objective optimization of hydrologic models. However, recent study [15] has shown that, in a set of standardized test functions and a couple hydrologic calibration cases, the MOSCEM algorithm is inferior to genetic algorithm-based multi-objective optimization algorithms, including NSGA-II (Nondominated Sorted Genetic Algorithm II) [16] and SPEA-2 (Strength Pareto Evolutionary Algorithm 2) [17], in measurement of both the capability of reaching Pareto optimal solutions and the computation time. One main reason is that the stochastic search operator in MOSCEM depends on the prior distribution, which requires covariance calculation of the parameters in the population. Due to this reason, the stochastic search in MOSCEM using complex shuffling "does not scale well for problems of increasing size and/or difficulty." [15] Moreover, the fitness assignment in MOSCEM requires consistently updating the current population, which is not amenable to the implementation on the emerging many-core parallel computing architecture.

In this paper, a Differential Evolutionary Markov Chain Monte Carlo (DE-MCMC) method for multi-objective optimization is presented as an improvement of the MOSCEM algorithm [14]. Differential Evolution [18], an efficient and robust evolutionary algorithm for continuous variables, is incorporated in a population-based MCMC sampler for optimization to evolve the Markov chains toward solutions at the Pareto optimal front in the presence of multiple objective functions. The DE operator does not rely on predefined probability distribution function; instead, DE uses the differences of the parameter vectors in randomly sampled individuals to generate offspring configurations, where the estimation of the prior distribution can be avoided. Moreover, the DE-MCMC algorithm is particularly suitable for the emerging Single Instruction Multiple Thread (SIMT) programming paradigm. The SIMT DE-MCMC algorithm is implemented on the Graphics Processing Unit (GPU) in the Compute Unified Device Architecture (CUDA) environment [19]. The DE-MCMC method is tested on a set of standardized test function suites as well as a protein loop structure sampling application [20] to demonstrate the effectiveness of the approach and the efficiency of the SIMT implementation.

The rest of the paper is organized as follows. Section 2 discusses the MCMC method for multi-objective optimization. Sections 3 and 4 illustrate the DE-MCMC algorithm and its implementation on GPU, respectively. Section 5 analyzes the computational results and Section 6 summarizes the paper with conclusions and future research directions.

2. MCMC FOR MULTI-OBJECTIVE OPTIMIZATION

In single objective optimization, suppose that we are interested in finding the global minimum of an objective function, $f(\mathbf{x})$, for all possible configurations $\mathbf{x}$. The fundamental formulation of MCMC is to make up a probability distribution:

$$\pi(\mathbf{x}) \propto e^{-f(\mathbf{x})/T}, T > 0,$$

where T is the simulated temperature. Then, a Markov chain is constructed to sample $\pi(\mathbf{x})$. The solutions in the vicinity of the global minimum of $f(\mathbf{x})$ are most likely to be drawn in the sampling process.

In multi-objective optimization, when N multiple objective functions, $f_1(\mathbf{x})$, …, $f_N(\mathbf{x})$, are presented, the probability distribution is converted to

$$\pi(\mathbf{x}) \propto e^{-dom(\mathbf{x})/T}, T > 0,$$

where $dom(\mathbf{x})$ is the dominance function. The definition of dominance function is based on the concept of Pareto dominance [21]. A configuration $\boldsymbol{u}$ is said to Pareto dominate another configuration $\boldsymbol{v}$ ($\mathbf{u} \prec \mathbf{v}$) if both following conditions i) and ii) are satisfied (assuming all objectives are for minimization):

i) for each objective function $f_i(.)$, $f_i(\boldsymbol{u}) \le f_i(\boldsymbol{v})$ holds for all i; and

ii) there is at least one scoring function $f_j(.)$ where $f_j(\boldsymbol{u}) < f_j(\boldsymbol{v})$ is satisfied.

Consequently, the dominance function $dom(\mathbf{x})$ is the integration of all possible configurations that dominate $\mathbf{x}$:

$$dom(\mathbf{x}) = \int_{y \prec x} 1 d\,|\,\mathbf{y}\,| \cdot$$

As a result, the MCMC for multi-objective optimization can be formulated as a sampling problem of the dominance function, where the Pareto optimal solutions correspond to the global minima of the dominance function with value 0.

In practice, since the Pareto optimal front is usually unknown, it is difficult to explicitly calculate the exact dominance function value for a configuration. Instead, a population of configuration samples is created and the Pareto dominance relationship is evaluated between each pair of the configurations, which can be used to estimate the dominance function value for each configuration. By sampling and minimizing the dominance function, the MCMC sampler evolves the population of Markov chains toward the solutions at the Pareto optimal front of the objective function space.

3. DE-MCMC ALGORITHM

3.1 Fitness Assignment

In many single objective optimization applications, the goal is to search for an optimal solution; whereas in multi-objective optimization, there are two equally important goals: 1) finding the Pareto optimal solutions; and 2) obtaining diversified coverage of the Pareto optimal front. Sampling the dominance function may result in clustering of solutions in certain regions of the objectives while missing the other Pareto optimal solutions. To preserve diversity in sampling, the dominance function is manipulated by giving preference to the undersampled solutions. This results in a new fitness function. Consequently, the MCMC sampler evolves based on the fitness function instead of the dominance function.

Our DE-MCMC algorithm employs the fitness assignment scheme [14] used in MOSCEM, which is based on to the number of external non-dominated points. Considering a population P with m individual configurations, C_1, …, C_m, the fitness calculation is based on the strength s_i of each non-dominated configuration C_i. The strength s_i is defined as the proportion of configurations in P dominated by C_i. Then, the fitness of an individual C_i is defined as

$$fit(C_i) = \begin{cases} s_i & C_i \text{ is non-dominated} \\ 1 + \sum_{j}^{i \succ j} s_j & C_i \text{ is dominated} \end{cases}.$$

The configurations with fitness less than 1.0 are non-dominated. The fitness function biases to the non-dominated ones with less dominated configurations while those with a lot of neighbors in their niche are penalized [14].

3.2 Differential Evolution

DE is used to produce new configurations based on the current population. In the DE scheme [18], for each configuration $C_i(x_1, \ldots, x_n)$ a mutant vector V_i is typically formed by

$$V_i = C_{r1} + F(C_{r2} - C_{r3})$$

where i, $r1$, $r2$, and $r3$ are mutually distinct integer random numbers in the interval [1, m] and $F > 0$ is a tunable amplification control constant as described in [18]. Other forms of mutant vector generation are also provided in [18], which can be employed in DE-MCMC as well. Then, a new conformation $C_i'(x_1', \ldots, x_n')$ is generated by the crossover operation on V_i and C_i:

$$x_j' = \begin{cases} v_j & j = \langle k \rangle_n, \langle k+1 \rangle_n, \ldots, \langle k+L-1 \rangle_n \\ x_j & otherwise \end{cases},$$

where $\langle . \rangle_n$ denotes the modulo operation with modulus n, k is a randomly generated integer from the interval [0, n-1], L is an integer drawn from [0, n-1] with probability $\Pr(L = l) = (\mathrm{CR})^l$, and $\mathrm{CR} \in [0, 1)$ is the crossover probability. Practical advice suggests that CR = 0.9 and F = 0.8 are suitable choices in the DE scheme [18]. These parameter values are also adopted in the DE-MCMC algorithm.

3.3 Metropolis Transitions

The Metropolis transition is employed as Monte Carlo moves for generating new population. The transition probability depends only on the change of the fitness function value. The Metropolis-Hastings ratio is calculated as:

$$w(C_i \to C_i') = e^{\frac{-(fit(C_i') - fit(C_i))}{T}}.$$

The new configuration C_i' generated by DE is accepted with the probability

$$\min(1, w(C_i \to C_i')) \cdot$$

T is the simulated temperature, which is used to control the acceptance rate of the Metropolis transitions. Studies [22, 23] in Monte Carlo method literature have shown that the MCMC sampler yields the best possible performance with an acceptance rate of around 20%. In this paper, we also maintain an acceptance rate of 15%~25% by adjusting T in each iteration. When the acceptance rate is less than 15%, T is increased by 10%. When the rate is more than 20%, T is decreased by 10%.

3.4 Description of DE-MCMC Algorithm

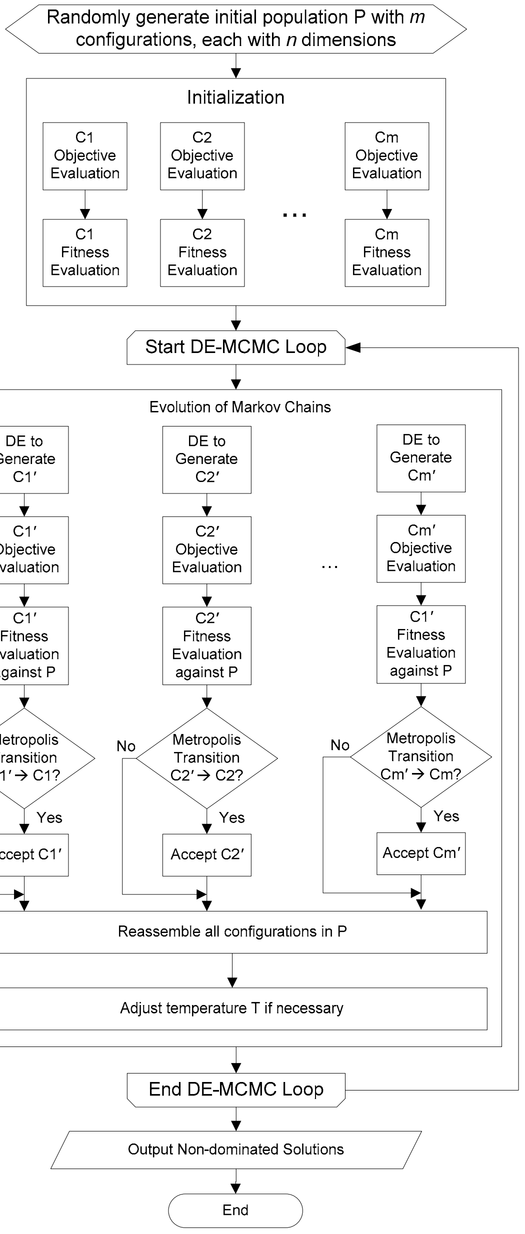

Randomly generate initial population P with m configurations, each with n dimensions
Initialization
C1 Objective Evaluation
C2 Objective Evaluation
Cm Objective Evaluation
C1 Fitness Evaluation
C2 Fitness Evaluation
Cm Fitness Evaluation
Start DE-MCMC Loop
Evolution of Markov Chains
DE to Generate C1′
DE to Generate C2′
DE to Generate Cm′
C1′ Objective Evaluation
C2′ Objective Evaluation
Cm′ Objective Evaluation
C1′ Fitness Evaluation against P
C2′ Fitness Evaluation against P
C1′ Fitness Evaluation against P
Metropolis Transition C1′→ C1?
Metropolis Transition C2′→ C2?
Metropolis Transition Cm′→ Cm?
No
Yes
Accept C1′
Accept C2′
Accept Cm′
Reassemble all configurations in P
Adjust temperature T if necessary
End DE-MCMC Loop
Output Non-dominated Solutions
End

Figure 1. Flowchart of DE-MCMC Algorithm

Putting every piece of the puzzle together, the flowchart of the DE-MCMC algorithm is shown in Figure 1. First of all, a population P of m configurations, C_1, …, C_m, are produced randomly and the corresponding objective functions values are evaluated. Then, the fitness function values of C_1, …, C_m are calculated against P. In the DE-MCMC loop, new configurations of C_1', …, C_m' are generated by DE. After evaluating the objective functions values of the new configurations, the fitness values of C_1', …, C_m' are calculated against the old population P. Afterwards, the Metropolis transitions are carried out to determine the acceptance of each new configuration. Then, the population P is updated by collecting the newly accepted configurations and replacing the old ones. The value of the simulated temperature T is adjusted to maintain a desired acceptance rate. The DE-MCMC computation reiterates until the expected iteration number is reached and then the non-dominated configurations are outputted as solutions.

Most computation in the DE-MCMC loop, including generation of new configurations, evaluation of objective functions, fitness assignment, and Metropolis transition, can be carried out independently and only require read operations to access the shared population information, which is particularly suitable for a massively parallel computing system with shared memory.

4. GPU IMPLEMENTATION OF DE-MCMC

The DE-MCMC algorithm was adapted for implementation on a CPU-GPU heterogeneous computing platform using the SIMT programming paradigm. The design target of SIMT DE-MCMC is to move heavily repetitive computation tasks to the GPU for data-parallel computing whereas to limit the CPU-GPU communication overhead. The SIMT DE-MCMC program is a combination of CPU and GPU codes. The initialization and evolution of Markov chains of the population P are carried out by m threads in the GPU, where each thread carries out the computation on a configuration.

4.1 Pseudorandom Numbers

One of the challenges in the CPU-GPU implementation of DE-MCMC is the requirement of a large number of parallel pseudorandom numbers. Unlike traditional CPU-based code, where pseudorandom numbers are typically generated on demand, for programs carried out in the GPU, a more efficient approach is to generate a large number of pseudorandom numbers before hand and then store them in the texture memory space to enable faster access by the kernels. In DE-MCMC, all pseudorandom numbers used in each iteration are produced at the beginning for all threads carried out in the GPU. In our implementation of DE-MCMC, a parallel Mersenne Twister pseudorandom number generator is adapted from the CUDA programming examples [19].

4.2 Objective Functions Evaluation and Fitness Assignment

The parallelization of objective functions evaluations is straightforward since they are independent in each thread. It will significantly save the overall computational time by parallelizing these computations. This is due to the fact that, in many optimization applications, most of the computational time is spent on objective function evaluation. The computational complexity of the objective function evaluation is $O(mN)$, where m is the population size and N is the number of objective functions.

The computation of the fitness assignment requires evaluation of dominance relationship between pair-wise configurations in population *P*. As a result, the computational complexity of evaluating the fitness of a population with size *m* and *N* objective functions is $O(m^2N)$. Therefore, the fitness assignment can be time-consuming when a large population is employed. The fitness assignment computation in each thread is also independent, which requires read operations to the shared objective functions values in the population.

4.3 GPU Tasks

Generation of new configurations using DE, pseudorandom number generations, objective functions evaluations, fitness assignment computations, and Metropolis transitions are carried out in the GPU in our DE-MCMC implementation. When the new configurations are generated in the global device memory in the GPU, they are copied into the texture memory for objective functions evaluation. After obtaining the values of every objective function, the results are copied to the texture memory for fitness assignment computation. Texture memory is used extensively to save time in accessing larger chunk of (possibly un-coalesced) memory that is more than the capacity of the available registers and shared memory. Coalesced read/write to the global memory is made possible by reorganizing the data.

5. COMPUTATIONAL RESULTS

5.1 Effectiveness of DE-MCMC in Multi-objective Optimization

5.1.1 A Simple 2D Optimization Problem

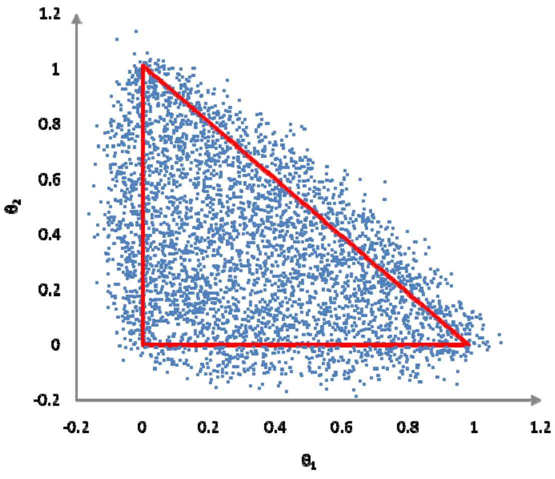


Figure 2. A scatter plot of the 4096 points in the solution space of the simple 2D optimization problem

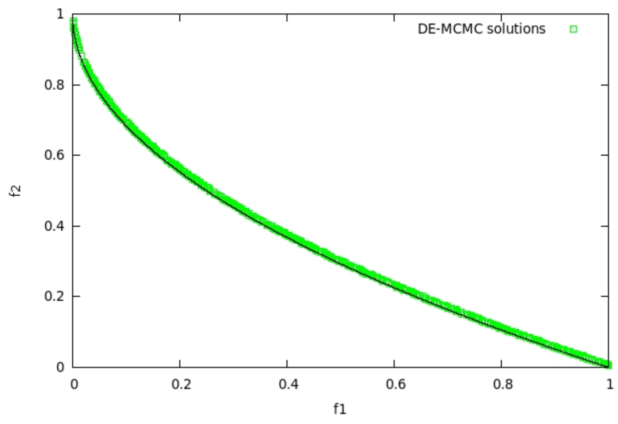
We tested the DE-MCMC algorithm on a simple mathematical test problem with two variables and three objective functions, which is used to show the effectiveness of MOSCEM in [14]. The test problem is described as

where the Pareto optimal solutions lay in the triangular-shaped area with the corner points (0, 0), (0, 1), and (1, 0) in the parameter space. Figure 2 shows a scatter plot of 4,096 solutions in the parameter space, where 3077 out of the 4096 solutions converge to the Pareto Optimal front after 100 iterations within less than one second using the GPU-accelerated program. DE-MCMC has demonstrated the uniform coverage of the solution space in this simple test problem, which is similar to that of MOSCEM.

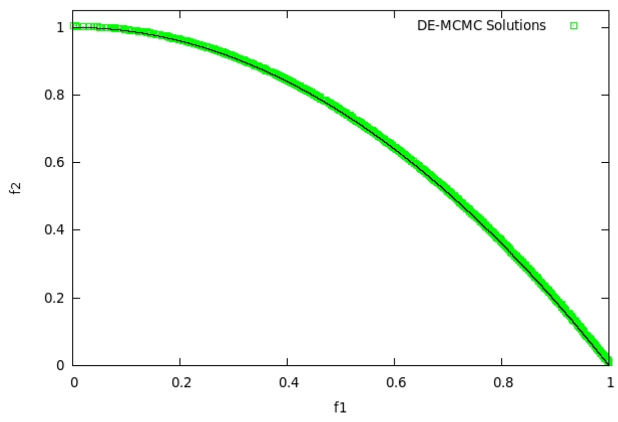
5.1.2 Test Functions Suites in PISA Library

We test the DE-MCMC algorithm on a set of standard test functions provided by the PISA library [30], which has been used broadly in multi-objective optimization literature. These test functions include ZDT1 (convex Pareto optimal front), ZDT2 (nonconvex Pareto optimal front), ZDT3 (Pareto optimal front with several noncontiguous convex parts), ZDT4 (Pareto optimal front with multimodality), and ZDT6 (nonuniform search space) [24][1], which have been used to study MOSCEM in recent study [15]. MOSCEM has been found to have unsatisfactory performance in these test functions which include poor convergence to the Pareto optimal front as well as requirement of significant amount of computational time (in magnitude of days).

Figure 3 shows the solutions found in DE-MCMC in ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6. A population size of 1,024 and 2,000 iterations are used in DE-MCMC. One can find that in all test functions, DE-MCMC yields good convergence and coverage of the Pareto optimal front. Moreover, with the help of GPU, the computation on a test case can be completed within a few seconds typically.

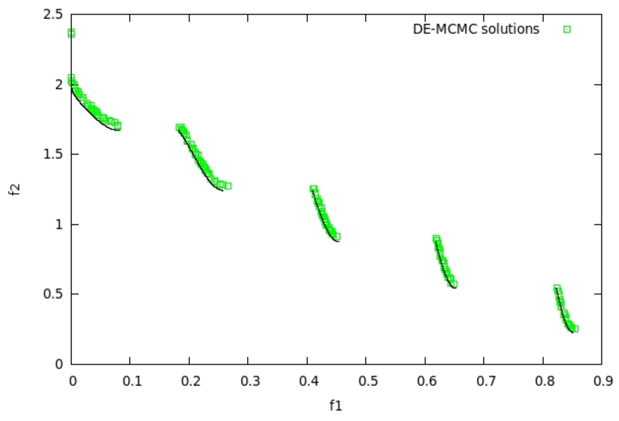


(a) ZDT1

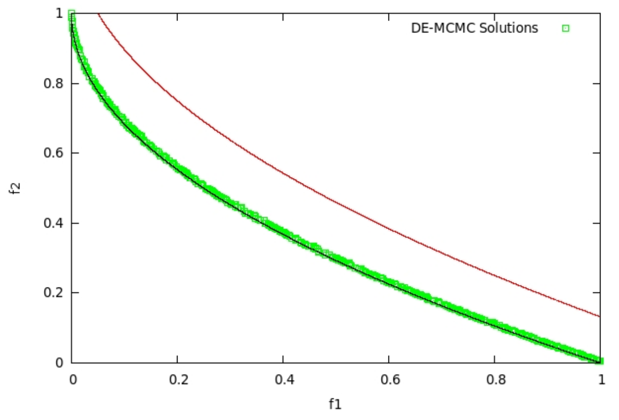


(b) ZDT2

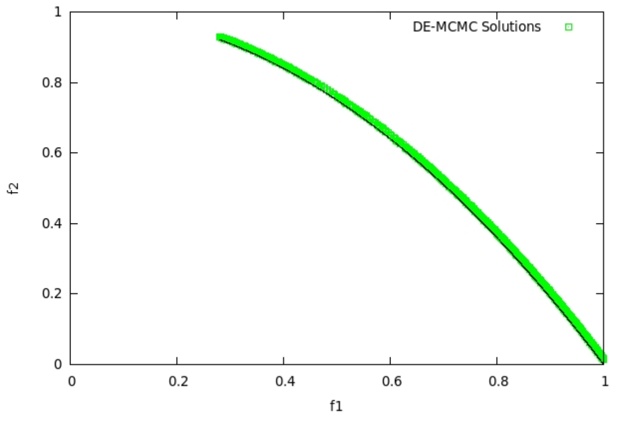
[1] ZDT5 is function on discrete space



(c) ZDT3



(d) ZDT4
(The red solid line represents a local Pareto optimal front)



(e) ZDT6

Figure 3. DE-MCMC solutions in test functions of ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6 (Population size of 1024, 2,000 iterations). The black solid lines represent the true Pareto optimal solution.

5.1.3 A Protein Structure Sampling Problem

We also applied the DE-MCMC algorithm to a protein loop structure sampling problem [20]. The rationale of the protein loop structure sampling is to obtain structurally diversified protein loop backbone conformations satisfying various scoring (objective) functions in order to discover conformations close to the native. Three scoring functions are used in this program, including a triplet torsion angle scoring function [25], an atom pair-wise distance-based scoring function [26], and a soft-sphere van der Waals energy function [27], which are used to measure the favorability of torsion angles, atom-atom interactions, and potential clashes, respectively. Generating new configuration is rather costly due to the protein loop closure requirement. Unlike the test functions suite where the computational time in objective function evaluation is small, the evaluations of the scoring functions and new configuration construction in the protein loop structure sampling program are rather costly. Together they occupy more than 95% of the overall computation time.

Figure 4 shows the structure-score distribution of 849 non-dominated solutions obtained in an 11-residue protein loop 153l(154:164) using the DE-MCMC algorithm with population size of 1,000 and 1,000 iterations. One can find that optimization using DE-MCMC has led to diversified, non-dominated solutions satisfying the three scoring functions. Among these solutions, a solution cluster close to the native conformation (< 0.5A) emerges.

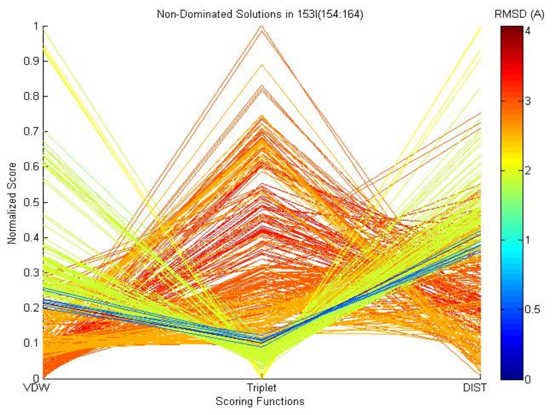


Figure 4. Structure-Score Distribution of 849 Non-dominated Solutions in Protein Loop 153l(154:164)

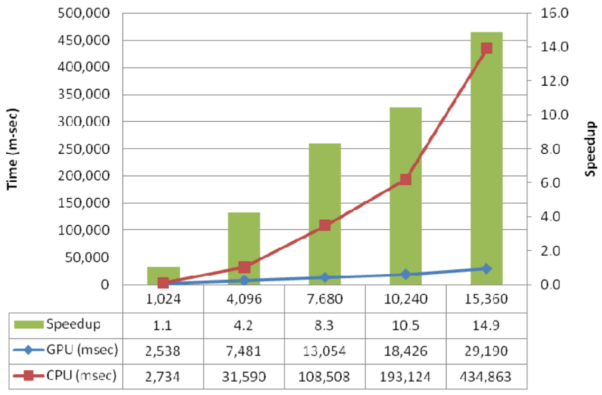
5.2 GPU Speedup Comparison on Benchmark Functions

Our SIMT DE-MCMC program is implemented in Visual C++ 2005 with the CUDA environment for programming the GPU. The same source code can also be compiled and run under Linux Ubuntu 9.04 with G++ 4.3 compiler. For benchmarking reason, the algorithm was also implemented with CPU-based only functions so as to compare the computation speed with the GPU-accelerated implementation. Both CPU and GPU implementations are tested on a Dell Precision 7400 Workstation computer with an Intel® Xeon™ E5420 CPU, 2GB memory, and an nVidia GeForce™ GTX 280 GPU with 240 (30 multi-processors x 8 processors per multi-processor) SIMT processors. The test functions adapted from PISA library have been selected for these computational experiments [30].

For management purpose, all threads carried out in the GPU are organized into blocks. In our computational experiments, we use a heuristic number of 256 threads per block. With 256 threads per block, we tested up to 15,360 threads. The number of blocks that can be launched by a GPU is determined by several factors, mostly the registers and/or shared memory needed per thread. When compiling the GPU kernels, it is possible to limit the number of registers used per thread. If the actual number of variables is more than the register number limit, then the variables will overflow to local memory, which is much slower and consequently affects performance. If the objective functions are very complicated, it is certain that the number of variables will be more than the available registers. Therefore, it is important to make judicious use of the memory.

Table 1. GPU speedup for a set of test functions (15,360 threads, 100 iterations, average of 20 runs)

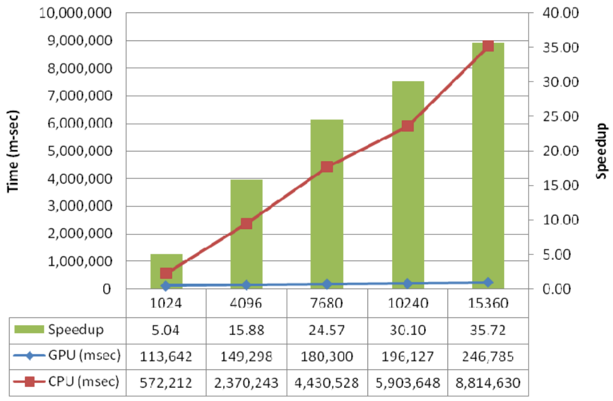
Function	Var	Obj	CPU (msec)	GPU (msec)	Speedup
Simple2D	2	3	368,832	22,618	16.3
ZDT1	100	2	332,713	20,824	16.0
ZDT2	100	2	358,484	21,591	16.6
ZDT3	100	2	331,468	20,825	15.9
ZDT4	100	2	371,681	20,090	18.5
ZDT6	100	2	361,576	22,109	16.4
SPHERE	100	3	422,519	22,746	18.6
KUR	100	2	400,108	20,374	19.6
QV	100	2	356,817	20,342	17.5
DTLZ1	100	3	434,863	29,190	14.9
DTLZ2	100	3	390,389	26,446	14.8
DTLZ3	100	3	395,186	26,315	15.0
DTLZ4	100	3	562,862	26,793	21.0
DTLZ5	100	3	413,112	27,686	14.9
DTLZ5	100	3	425,273	29,503	14.4
DTLZ7	100	3	442,734	30,350	14.6
COMET	3	3	380,824	24,685	15.4



	1,024	4,096	7,680	10,240	15,360
🟩 Speedup	1.1	4.2	8.3	10.5	14.9
GPU (msec)	2,538	7,481	13,054	18,426	29,190
CPU (msec)	2,734	31,590	108,508	193,124	434,863

Figure 5. GPU speedup comparison based on the number of threads in DTLZ1 test function (100 variables, 3 objectives, 100 iterations, average of 20 runs)

Table 1 summarizes the GPU speedup on a set of scalable test functions adapted from the PISA library with 15,360 threads [30]. One can find that the speedup is around 14~21. Figures 5 and 6 illustrate the GPU speedup comparison based on the number of threads in DTLZ1 test function and protein loop sampling program, respectively. It is interesting to find that the computational time in the CPU program increases quadratically as the number of threads increases in the DTLZ1 test function while in the protein loop sampling program, the increase is nearly linear. More importantly, the GPU implementation of DE-MCMC for protein loop structure sampling leads to more significant speedup – when 15,360 threads are in use, the GPU speedup can reach 35. This is due to the fact that in the simple test functions, evaluation of the objective functions is fast and most of the computation time of the program is in fitness assignment as shown in Table 2, whose computation time increases quadratically as the number of threads (population size) increases. In contrast, Table 3 shows that the majority computation time in protein loop structure sampling is spent on evaluation of the objective (scoring) functions and new configurations generation while the fitness assignment time is relatively small in comparison, which leads to nearly-linear computation time increase for larger population size. Furthermore, parallelizing the costly objective function evaluation in GPU can efficiently utilize resources in GPU and reduce the percentage of synchronization overhead in each DE-MCMC iteration, which leads to more significant speedup in protein loop structure sampling problem than those in simple test functions.



	1024	4096	7680	10240	15360
🟩 Speedup	5.04	15.88	24.57	30.10	35.72
GPU (msec)	113,642	149,298	180,300	196,127	246,785
CPU (msec)	572,212	2,370,243	4,430,528	5,903,648	8,814,630

Figure 6. GPU speedup comparison based on the number of threads in protein loop sampling program on a 12-residue loop 1xyz(813:824)

Table 2. Time breakdown of GPU tasks for DE-MCMC in DTLZ1 function with 100 variables and 3 objectives (15,360 threads, 100 iterations)

Category	Method	#calls	GPU (usec)	GPU time %
kernel	fitness assignment	100	20,129,500	90.20%
kernel	metropolis	99	711,078	3.18%
kernel	make new solutions	99	506,476	2.26%
kernel	Random # Generation	99	57,930	0.25%
kernel	objectives evaluation	100	20,966	0.09%
Mem sync	memcpyHtoD	307	381,520	1.70%
Mem sync	memcpyDtoA	497	76,862	0.34%
Mem sync	memcpyDtoH	299	427,729	1.91%
Mem sync	memcpyHtoA	100	3,547	0.01%

Table 3. Time breakdown of GPU tasks for DE-MCMC in protein loop structure sampling program (15,360 threads, 1xyz(813:824), 100 iterations)

Category	Method	#calls	GPU (usec)	GPU time %
kernels	make new configurations	101	166,480,692	71.06%
kernels	objectives evaluation	101	58,070,000	24.77%
kernels	fitness assignment	101	7,736,722	3.29%
Mem sync	memcpyHtoA	10	72,467	0.03%
Mem sync	memcpyHtoD	518	93,472	0.03%
Mem sync	memcpyDtoA	202	277,701	0.11%
Mem sync	memcpyDtoH	706	1,520,560	0.64%
Mem sync	memcpyDtoD	303	1,851	0.00%

5.3 Impact of Population Size

Unlike most single objective function optimization applications, where there are usually one or several solutions at the global minimum, the goal of multi-objective functions optimization is to sufficiently sample the solutions at the Pareto optimal front. As a result, the population size is a critical parameter in DE-MCMC, which depends on the complexity of the problem. Typically, in a complex multi-objective optimization problem, a larger population size will lead to a better coverage of the Pareto optimal front. Figure 7(a) shows that using a small population size (256) in DE-MCMC leads to undersampling or gaps in the solutions covering the Pareto optimal front of the ZDT4 test function. Similarly, Figure 8(a) shows that DE-MCMC with a small population size (256) causes missing a Pareto optimal segment in a modified ZDT3 test function (n = 30) described as,

$$f_1(\boldsymbol{x}) = x_1$$

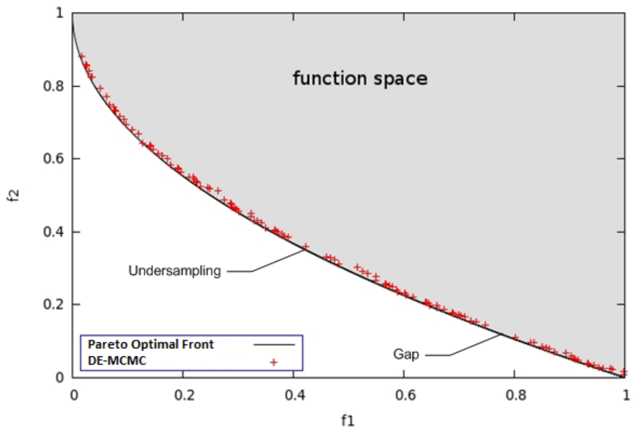
$$g(x_2, \dots, x_n) = 1 + 9 \cdot \sum_{i=2}^{n} \frac{x_i}{n-1}$$

$$h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}} - (f_1/g)\sin(50\pi f_1)$$

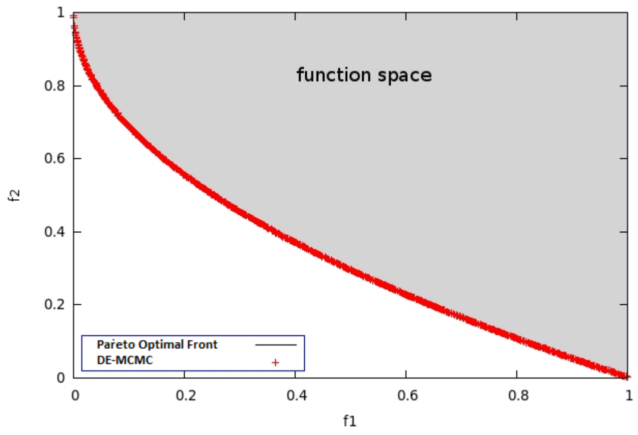
$$f_2(\boldsymbol{x}) = g \cdot h(f_1, g)$$

which has 25 separated Pareto optimal fronts. In contrast, both Figures 7(b) and 8(b) show good coverage of the corresponding Pareto optimal front when a sufficiently large population size (4096) is used in DE-MCMC.

It is also important to notice that in Figures 5 and 6, the CPU implementation of DE-MCMC using population size of 15,360 demands ~159 times and ~15 times more computational time than the program with population size of 1,024 in DTLZ1 test function and protein loop structure sampling problem, respective. In contrast, the computational time in CPU-GPU implementation of DE-MCMC using population size 15,360 is only ~11.5 and ~2.2 times of the one with population size 1,024 in these two test cases. This indicates that the heterogeneous CPU-GPU architecture is cost-efficient in carrying out the DE-MCMC algorithm with large population size.

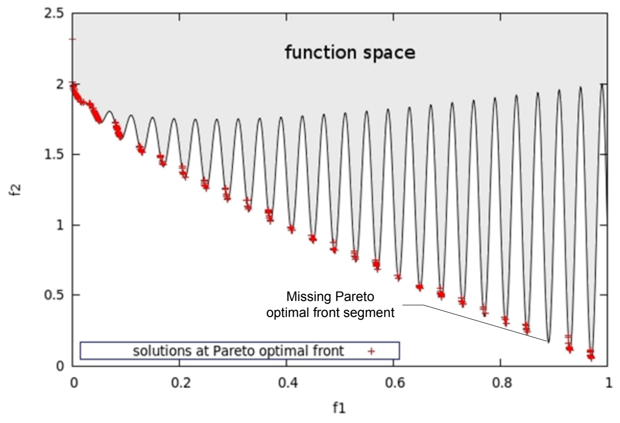


(a) Population Size of 256

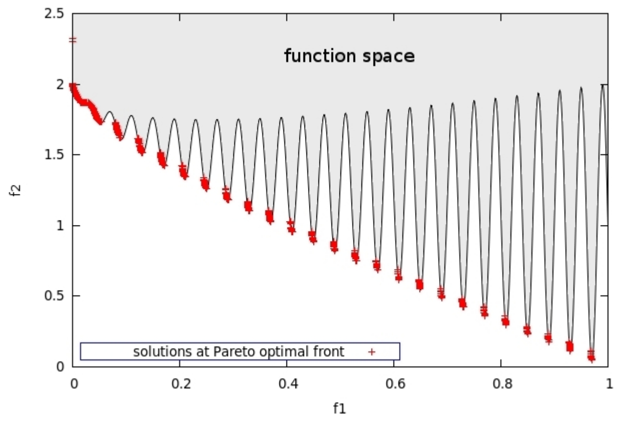


(b) Population Size of 4096

Figure 7: A small population size (256) leads to solution undersampling and gaps in the Pareto optimal front of ZDT4 while a large population size (4096) can have sufficient coverage of the Pareto optimal front



(a) Population Size of 256



(b) Population Size of 4096

Figure 8: A small population size (256) causes missing a Pareto optimal segment in a modified ZDT3 test function with 25 separated Pareto optimal fronts while a large population size (4096) can lead to good sampling of Pareto optimal solutions

6. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

In this paper, we present a new multi-objective optimization approach by incorporating DE, an evolutionary scheme, into a population-based MCMC sampler. The DE-MCMC method has demonstrated its effectiveness in a set of test functions, where the MOSCEM algorithm has shown certain deficiency, as well as a protein loop structure sampling problem. Moreover, the DE-MCMC method can be adopted in the SIMT programming paradigm and efficiently take advantage of the emerging many-core architectures such as the GPU.

Although there are a number of existing genetic algorithm-based multi-objective optimization methods, one of the key advantages of MCMC is that the statistics of the Markov chains can be used to determine the important parameters of the algorithm as well as to characterize the optimization process. In our future research, we plan to take advantage of the MCMC analysis tools, such as mixing time [28] and autocorrelation function [29], to study the DE-MCMC algorithm for multi-objective optimization. We expect these tools will be able to help us determine the appropriate population for a particular multi-objective optimization problem and the convergence of optimization process.

7. ACKNOWLEDGEMENT

The work is partially supported by NSF under grants CCF-0829382 and CCF-0845702 to Y. Li and Lamar Research Enhancement Grants, a grant from Texas Hazardous Waste Research Center, and NSF grant DUE-0737173 to W. Zhu.

8. REFERENCES

[1] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, "Equation of State Calculations by Fast Computing Machines", Journal of Chemical Physics, **21**:1087-1092, 1953.

[2] W. K. Hastings, "Monte Carlo Sampling Methods using Markov Chains and Their Applications," Biometrika, **57**:97-109, 1970.

[3] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, "Optimization by Simulated Annealing," Science, **220**:671-680, 1983.

[4] E. Marinari and G. Parisi, "Simulated Tempering: a New Monte Carlo Scheme," Europhysics Letters, **19**:451-458, 1992.

[5] C. J. Geyer and E. A. Thompson, "Annealing Markov Chain Monte Carlo with Applications to Ancestral Inference," J. of the American Statistical Association, **90**:909-920, 1995.

[6] U. H. E. Hansmann, Y. Okamoto, "Generalized-ensemble Monte Carlo method for systems with rough energy landscape," Physical Review E., **56**(20): 2228-2233, 1997.

[7] J. S. Liu, F. Liang, and W. H. Wong, "The Use of Multiple-Try Method and Local Optimization in Metropolis Sampling," Technical Report, Department of Statistics, Stanford University, 1998.

[8] Y. Li, V. A. Protopopescu, N. Arnold, X. Zhang, A. Gorin, "Hybrid Parallel Tempering/Simulated Annealing Method," Applied Mathematics and Computation, **212**:216-228, 2009

[9] J. I. Siepmann, D. Frenkel, "Configuration bias Monte Carlo: a new sampling scheme for flexible chains," Molecular Physics, **75**(1):59-70, 1992.

[10] M. E. J. Newman, G. T. Barkema, "Monte Carlo Methods in Statistical Physics," Oxford University Press, 1999.

[11] M. Laine, J. Tamminen, "Aerosol model selection and uncertainty modeling by adaptive MCMC technique," Atmos. Chem. Phys., **8**: 7697-7707, 2008.

[12] S. Wang, S. J. Mitchell, P. A. Rikvold, "Ab initio Monte Carlo simulations for nanoscopic lithium systems at different temperatures," Comput. Mat. Sci. **29**:145, 2004.

[13] Z. Li and H. A. Scheraga, "Monte Carlo-Minimization Approach to the Multiple-Minima Problem in Protein Folding," Proceedings of the National Academy of Sciences of the USA, **84**(19):6611-6615, 1987.

[14] J. A. Vrugt, H. V. Gupta, L. A. Bastidas, W. Boutem, S. Sorooshian, "Effective and Efficient Algorithm for Multiobjective Optimization of Hydrologic Models," Water Resources Research, **39**(8): 1214-1232, 2002.

[15] Y. Tang, P. Reed, T. Wagener, "How effective and efficient are multiobjective evolutionary algorithms at hydrologic model calibration?" Hydrol. Earth Syst. Sci., **10**: 289-307, 2006.

[16] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," IEEE Trans. Evol. Computation, **6**, 182-197, 2002.

[17] E. Zitzler, M. Laumanns, L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," TIK-103, Department of Electrical Engineering, Swiss Federal Institute of Technology, Zurich, Switzerland, 2001.

[18] R. Storn, K. Price, "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," J. of Global Optimization, **11**(4):341–359, 1997.

[19] nVidia, 2010, CUDA Programming Guide Version 2.3. Available at: http://www.nvidia.com/object/cuda_get.html.

[20] Y. Li, I. Rata, E. Jakobsson, "Multi-Scoring Functions Sampling in Protein Loop Structure Prediction," submitted to Applied Mathematics and Computation, 2009.

[21] K. Deb, "Multi-objective optimization using evolutionary algorithms," John Wiley&Sons, 2001.

[22] N. Rathore, M. Chopra, J. J. de Pablo, "Optimal Allocation of Replicas in Parallel Tempering Simulations," J. Chem. Phys., **122**(2):024111, 2005.

[23] A. Kone, D. A. Kofke, "Selection of Temperature Intervals for Parallel Tempering Simulations," J. Chem. Phys., **122**(20): 206101, 2005.

[24] E. Zitzler, K. Deb, L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," Evolutionary Computation, **8**(2): 173-195, 2000.

[25] A. Rojnuckarin, S. Subramaniam, "Knowledge-based interaction potentials for proteins", Proteins: Structure, Function, and Genetics **36**: 54-67, 1999.

[26] I. Rata, Y. Li, E. Jakobsson, "Backbone statistical potential from local sequence-structure interactions in protein loops," Journal of Phys. Chem. B., **114**(5): 1859-1869, 2010.

[27] H. Zhang, L. Lai, Y. Han, Y. Tang, "A Fast and Efficient Program for Modeling Protein Loops," Biopolymers, **41**: 61-72, 1997.

[28] D. A. Levin, Y. Peres, E. L. Wilmer, "Markov Chains and Mixing Times," American Mathematical Society, Providence, RI, 2008.

[29] J. Goodman, A. Sokal, "Multigrid Monte Carlo Method: Conceptual foundations," Physical Review D, **40**(6): 2035-2071, 1989.

[30] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler, "PISA – A Platform and Programming Language Independent Interface for Search Algorithms," LNCS, **2632**: 494-508, 2003.