Surrogate-Based Multiobjective Optimization: ParEGO Update and Test

Cristina Cristescu
School of Computer Science
University of Manchester
Manchester M13 9PL
Email: bianca.cristescu@student.manchester.ac.uk

Joshua Knowles
School of Computer Science
University of Birmingham
Edgbaston B15 2TT
Email: j.knowles.1@cs.bham.ac.uk

Abstract—We consider ParEGO, a well-known algorithm in the Evolutionary Multiobjective Optimization (EMO) community, and make improvements to the implementation itself and to some of the underlying algorithm components. ParEGO is a surrogate-based multiobjective optimization algorithm based on the Kriging/DACE model, designed specifically for problems with expensive evaluation functions. First described in 2006, it has been successful in empirical comparison studies and real-world applications (e.g., in water network design and iterative experimentation) since its release, but its code relies on now outdated matrix routines, which we here replace by more modern libraries. The code is also ported to GPU for further acceleration. We also add options to the ParEGO algorithm itself: Originally limited to runs of up to about 250 function evaluations (due to the heavy computational cost of some matrix operations on the full search trace), we now include methods for discarding (or ‘forgetting’) selected previous search points. We find that the pure implementation updates reproduce the performance of the original ParEGO release across a suite of benchmark problems; further, the algorithmic changes enable scaling up of ParEGO runs (i.e., no. generations) without significant loss of accuracy of the model in tests of up to 500 evaluations. The project’s code is now available for use by the EMO community and its uptake in applications with expensive functions is encouraged.

I. INTRODUCTION

While EAs have proven powerful for a wide variety of optimisation problems, they generally require a large number of solution evaluations, which is detrimental in the case of functions that are expensive. For this context, where the evaluations are expensive or their number is restricted, an increasingly typical approach is to use surrogate (or meta-) models of the search landscape, updated at runtime, in order to replace the expensive evaluation of some candidate search points by a cheaper alternative. Mockus is usually cited as originating the idea [1], with Stewart contributing early related analyses [2]. More recent reviews of this area and its intersection with Computational Intelligence methods are given in [3] and [4]. Approaches developed for the multiobjective optimization case date back to [5], and related papers of the era are by Keane (e.g. see [6]) and Emmerich et al. [7].

In 2005, Knowles proposed in [8], [9] an algorithm which translates the successful EGO algorithm [10] into a multiobjective version given the name ParEGO (for Pareto EGO).

The characteristics of the problems that ParEGO aims to solve, presented by Knowles in [9], are the following: “

1) The problem has multiple, possibly incommensurable, objectives.
2) The time taken to perform one evaluation is of the order of minutes or hours.
3) Only one evaluation can be performed at one time (no parallelism is possible).
4) The total number of evaluations to be performed is limited by financial, time or resource constraints.
5) No realistic simulator or other method of approximating the full evaluation is readily available.
6) Noise is low (repeated evaluations yield very similar results).
7) The overall gains in quality (or reductions in cost) that can be achieved are high.
8) The search landscape is locally smooth but multimodal.
9) The dimensionality of the search space is low-to-medium.”

The motivation for developing such a method comes from the high potential for applications in real-life scenarios. This is exemplified with an application of ParEGO in analytical chemistry, optimizing a gas chromatography/mass spectrometry (GC-MS) instrument for serum metabolomics [11], and one in engineering, in urban wastewater systems [12].

In this paper, we describe a project to update the original ParEGO software, considering both critical changes to the implementation including a port to the GPU, and also additional (optional) changes to the algorithm. Section II reviews the main steps of the ParEGO algorithm, while Section III describes a new option implementing several methods to discard selected candidate points from the model updates. Section IV gives details of the changes (especially to the matrix library) in a new release of the code. The resulting algorithm and release is tested in Section V on a suite of eight benchmark problems from the original ParEGO paper. We find that implementation changes reproduce the performance of the original ParEGO release, while the new option enables the method to be used for more iterations (i.e. longer search runs). Concluding discussions and ideas for future development can be found in Section VI and VII.

1 The version of ParEGO described here can be downloaded at https://github.com/CristinaCristescu/ParEGO_Eigen
II. PAREGO BASICS

The Efficient Global Optimization method [10] is an iterative optimization method that uses surrogate modeling to approximate the real (expensive) objective function, and exploits the (iteratively updated) model to select subsequent sample points. The approximation of the landscape, also known as a response surface, can be used for visualization as well as optimization.

EGO uses a stochastic process model — a Gaussian Process, a.k.a. a Kriging model. The specific process employed, known as the Design and Analysis of Computer Experiments (DACE) model, has several advantages, among them: (i) The likelihood of the model given the data has a simple closed form expression from which it is possible to compute the maximum-likelihood model giving the approach a sound statistical interpretation, and (ii) The error in the expected cost of a solution also has a simple, closed form expression. Thus, the model estimates its own uncertainty. Since its only main disadvantage is that it uses expensive matrix inverse operations, which are not so detrimental in expensive applications, EGO has had a strong following.

To extend the single objective algorithm to cater for multiple objective problems, Knowles proposed using the weighted scalarizing function. As described in [9], ParEGO begins by normalising the objective functions with respect to the decision space. At each iteration, a weight vector is chosen randomly. The scalar cost is computed using the Tchebycheff augmented function in equation (1):

$$f(x) = \max_{j=1}^{k} (\Lambda_j f_j(x)) + \rho \sum_{j=1}^{k} \Lambda_j f_j(x) j \in 1..k$$  \hspace{1cm} (1)

where $\Lambda$ is the weight vector for each of the $k$ cost functions and $\rho$ is a small positive value set to 0.05.

Using the scalar cost of all or a selection of the previously visited solutions, the DACE model is constructed and the solution that maximises the expected improvement (see [10]) is obtained using a genetic algorithm. The resulting solution is the next to be evaluated on the real-expensive function and used to update the DACE model, completing one iteration of the algorithm. Fig. 1 illustrates the stages in the algorithm.

III. ALGORITHM UPDATES

The improvements of ParEGO suggest using a smaller set of previously found solutions to update the DACE model. At each iteration of ParEGO, the DACE model is recomputed. This involves a large number of costly matrix operations for which the overhead becomes significant. Using only part of the previously obtained solutions to update the model gives more weight to newer data in order to obtain more accurate results in fewer iterations. This is similar to the online learning technique. The maximum size of the matrix used to recompute the DACE model parameterises the shrinking process. Following this approach experiments have been done with the usage of only most recent solutions, most fit solutions and random solutions.

A. Recent Solutions

This heuristic relies on removing old data points from consideration. This has been implemented by adapting the online learning techniques of the moving window approach [13]. Thus, from the current populations of solutions, only a limited number of solutions will be considered. Since the solutions are stored in the order they have been found, extracting an interval of solutions is intuitively done by considering a ‘moving window’ of solutions.

B. Fittest Solutions

This heuristic aims to provide only the most fit solutions for the model to be updated. To implement this the current set of solutions is sorted in respect to their fitness, and the top fittest solutions are used (e.g. 150 solutions out of 200).

C. Random Solutions

Stochastic processes have proven very efficient, in particular in optimisation, as exemplified by the Random Walk technique. This heuristic chooses at random which points to use from the set of solutions chosen already. The undertaken statistical technique is k-sampling without replacement.
IV. IMPLEMENTATION UPDATES

The implementation updates considered fall into two categories: a modular implementation of ParEGO and a hybrid CPU-GPU implementation.

A. Modular ParEGO

The language choice for re-implementation is C++ due to speed constraints given by expensive matrix operations within the Gaussian process and availability of linear algebra libraries. When choosing amongst the existing linear algebra libraries the criteria was maintainability and time efficiency for matrix operations. The considered libraries were: ALGLIB, Armadillo, Eigen, MKL and GNU Scientific Library. The chosen library, Eigen, is used in many projects, has many contributors, and reliable sponsorship. In addition, it is highly optimised for matrix operations. Replacing the usage of the original linear algebra library, Matpack, by Eigen, required restructuring of the algorithm. The modular implementation posed difficulty in the decoupling process. Firstly, there is a high level of communication (e.g. the Genetic Algorithms uses the expected improvement computed in the DACE model) among the different concepts present in the composition of the algorithm. Implementing the communication between the classes, required several sub-iterations to obtain an efficient and easily maintainable application.

The flow shown in Figure 1, as outlined in one of the applications of ParEGO in [12], is followed. Therefore, the controller class ParEGO starts the initialisation of the algorithm. Having chosen a problem to optimise, the formal expression of a multi-objective optimisation is described by the SearchSpace class. After the parameters of the problem have been set, the initial solution population is created using the latin hypercube sampling method, provided in the Utilities class. This is in turn used by a DACE object to create a metamodel for the real-expensive function. The WeightVector class aggregates the objectives, using the Tchebycheff function, and the model is fitted using a method available in the DACE class. Then, the population is evolved using the Genetic Algorithm. The Genetic Algorithm performs its operations on the population and communicates with the DACE object to obtain an evaluation of the fitness of an individual, in terms of the expected improvement, using a function member of the DACE class. Subsequently, the DACE model obtained is re-evaluated and rebuilt taking into account the newly evolved solution. The new population is evaluated and a new iteration is started.

The modular implementation of ParEGO lends itself to the addition of a Wrapper Matrix class. Preliminary testing has been done to ensure that the overhead, introduced by using a wrapper, would not be significant. The undertaken test has been done using a couple of operations: addition, subtraction and multiplication. The results plotted in Figure 2 show that the overhead is negligible, on most matrix dimension, and the difference is barely visible, encouraging the use of the wrapper.

B. CPU-GPU Implementation

Recent studies [14], [15] have been conducted on the use of GPU computational power where high computational resources are required in MOEA (Multi-Objective Evolutionary Algorithms) in order to perform computation in parallel. In our particular case, the theoretical worst-case complexity of the algorithm is O(n^3), since the dominant operation is matrix inversion. Improvements to the performance of matrix inversion through a mathematical technique, called LU (lower-upper) factorisation, which decomposes a matrix as the product of a lower triangular matrix and an upper triangular matrix.

Providing a hybrid implementation which runs on the CPU, but enables expensive linear algebra operations to be performed on the GPU, is motivated by the time profile of the running ParEGO application, which reports more than 50% time spent on matrix operations. The challenge here was in justifying the decision in respect to the tools and technologies used, since the field of GPU programming is a young one. Several materials, which show that running linear algebra on the GPU has proven efficient, have been consulted. The reference paper was the reflection of Jack Dongarra, one of the pioneers of linear algebra implementations in software, on the existing solutions for dense linear algebra operations in [15]. Given that GPUs are inherently useful for vectorisation [16], [17] they are conceptually a good choice for ParEGO.

The benefits of the Matrix Wrapper Class were seen in the isolation of the changes needed be be done to the wrapper class, MyMatrix, in order to hybridise the application. Selected operations have been provided with a GPU version of the code using C++ pragmas\(^2\), which enables the two different versions to be compiled using different compilers i.e. gcc and nvcc. At compile time, the use of the GPU, referred to as the device, can be decided, and the corresponding code path taken.

The GPU specific operations are written in a dedicated class, Cuda_Utilsities. In order to perform the operations on

\(^2\)Compiler-specific preprocessor extensions
Fig. 3. The Hypervolume Indicator measurement for the selection of solutions to be used when updating the DACE model. It shows that there is a slight decrease in quality with three all methods: least fit, recent and random solutions to be ignored.

Fig. 4. The time performance measurement for the selection of solutions to be used when updating the DACE model. It shows time improvement with all methods used: least fit, recent and random solutions to be ignored.
the GPU, the data is copied to the memory of the device. The parameters of the operation to be performed are set, and the operations run in parallel on the GPU. The result is then collected from the memory of the device and feed over to the MyMatrix object. Another performance improvement was obtained by performing operations for small matrices, under a specific threshold (matrix size 50), on the CPU, avoiding the overhead of copying them to the device memory.

V. Evaluation

A running updated ParEGO algorithm, with different specifications in respect to performance, as detailed in the previous sections, was obtained. The algorithm has nine test functions within specified. These functions have different properties and pose different difficulties to ParEGO, and MOEA in general, as summarised in Table II. The user interface enables the choice of the following three parameters, as illustrated in Table III. The result of a run of the algorithm is an approximation set to the real Pareto front which consists of all non-dominated solutions found during the run. The output describes both the decision points and the objective values.

<table>
<thead>
<tr>
<th>Test function</th>
<th>Objective space dim., k</th>
<th>Decision space dim., n</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNO1</td>
<td>2</td>
<td>2</td>
<td>“The true Pareto front lies just beyond a locally optimum Pareto front with a much larger basin of attraction”</td>
</tr>
<tr>
<td>OKA1</td>
<td>2</td>
<td>2</td>
<td>“The density of the solutions fall away near to the Pareto front”</td>
</tr>
<tr>
<td>OKA2</td>
<td>2</td>
<td>3</td>
<td>“Pareto optima lie on a spiral-shaped curve and the density of the solutions falls away steeply near to the Pareto front”</td>
</tr>
<tr>
<td>VLMOP2</td>
<td>2</td>
<td>2</td>
<td>Pareto optimal set is disconnected and the Pareto front is a curve following a “convoluted path through objective space”</td>
</tr>
<tr>
<td>VLMOP3</td>
<td>3</td>
<td>2</td>
<td>Pareto front is concave</td>
</tr>
<tr>
<td>DTLZ1a</td>
<td>2</td>
<td>6</td>
<td>Increased level of function ruggedness</td>
</tr>
<tr>
<td>DTLZ2a</td>
<td>3</td>
<td>8</td>
<td>Pareto front is a sphere</td>
</tr>
</tbody>
</table>

ParEGO is a stochastic optimisation method which implies that testing cannot be performed in a traditional manner. As stated above, the result of the algorithm is an approximation set and the evaluation relies on “some means of measuring or comparing the quality of one or more approximation sets” [9]. The selected performance analysis techniques are the popular evaluation techniques used in MOO, Hypervolume Indicator and Attainment Surfaces plotting.

1) Shrinking the Update Matrix vs Full Update Matrix: Shrinking the matrix used to rebuild the DACE model is a tradeoff between the quality of the results and the time performance obtained. The maximal matrix size parameter, presented above, was used to carry out the experiments, i.e. the matrix used to update the DACE model was capped to a maximal size of 50, 100, 150, 200 respectively. The quality results are illustrated using the Hypervolume Indicator measurement in Figure 3 and the time performance plot is shown in Figure 4. The three methods employed in selecting the solutions used in the matrix updating the DACE model show a slight decrease, but this is acceptable given the time performance improvement obtained. Out of the three methods, selecting the fittest solutions is the best option followed by selecting them at random, and least effective is selecting the most recent solutions. The results are compiled using the mean hypervolume achieved by each of methods of shrinking the update matrix vs using the full update matrix (labelled here as ‘Eigen’); the highest value for each function is shaded. Data are from twenty independent runs. A multiple comparison test, Tukey’s HSD, was used to test differences between the methods. Although there were differences, no significant difference was found between Eigen and the method of discarding the least fit solutions as shown in Table I.

2) Modular ParEGO vs Original ParEGO: Evaluation of the modular version of ParEGO is done by comparing it to the
The Eigen-based modular ParEGO is statistically compared against the original MATPACK-based version. The signed difference in the mean hypervolume achieved from 20 independent runs of each method, on each function in turn, is given. The p-value quoted is the two-tailed t-test corrected for unequal variances. Taking a standard alpha value of 0.05 and correcting for the multiple test (nine different functions), we should reject the null hypothesis (of no difference in means) if $p < 0.0056$. There is no evidence to suggest a difference in performance except for the result on the function KNO1. The results are summarised in the BoxPlots in Figures 5 and 6. The results were expected to be the same since the algorithm is conceptually the same and the two versions were run for the same seed of the random numbers, and on the same machine.

The second evaluation was plotting Worst Attainment Surface Plot. It was expected that the Eigen version should perform at least as good as the Matpack version, and the expectation was fulfilled. This exemplified for one of the test functions DTLZ2a in Figure 7 showing the attainment for the version with Matpack which is similar to the attainment surface obtained using the new version of the algorithm in Figure 8.

3) CPU vs CPU-GPU: The CPU-GPU implementation is expected to have a speed-up related to the number of floating points operations that can be performed in a second (GFLOPS). Since the number of GFLOPS is linearly increasing from the CPU to the GPU as showed by the developers at NVIDIA [19] (illustrated in Figure 9) and considering the average-case complexity and worst-case complexity of the algorithm, we might hope for a near linear speed-up. The plot in Figure 10 does not show this occurring however. The following two points are a pertinent explanation for what is happening. Firstly, there is an overhead in the transfer of the matrices to and from the memory of the GPU which has not been addressed properly, and secondly, the Genetic Algorithm can create a bottleneck which does not fully allow the parallelisation provided by the GPU to be exploited.

VI. FURTHER PLANNED UPDATES

A further update to the algorithm is planned to improve the diversity in the approximation set, and two methods have been considered good candidates for future work. Although, ParEGO’s use of wide spread scalarising vectors encourages diversity to some extent, it does not adaptively pursue solutions in sparse areas of the current PF approximation. We have made steps to integrating two diversity schemes from the literature: that used in Pareto Simulated Annealing [20], and the Cone $\varepsilon$-Dominance [21] which was developed since the design of ParEGO.

VII. CONCLUSIONS

We have provided an updated and improved implementation of the ParEGO algorithm (first proposed in 2005) which offers the same functionality as before, with at least the same performance results. The new implementation is more modular in its code, following general software engineering principles, and the Eigen linear algebra library has been integrated as an appropriate and supported replacement to the original but outdated linear algebra library, Matpack. Moreover, additional experimentation with CPU-GPU hybrid architectures, and with improvements to the algorithm itself, provided interesting results for improving the performance of the algorithm.

Further, one of the potential drawbacks of ParEGO mentioned in the original paper [9] was the simple method of selecting scalarizing weight vectors (at random), which will be
Fig. 9. Number of GFLOPS is linearly increasing from the CPU to the GPU as claimed by the developers at NVIDIA [19].

Fig. 10. GPU speed-up.

addressed shortly in a further development of the algorithm.

Another future task now planned is to extend the GPU implementation; this should provide a distributed genetic algorithm to balance the work for one iteration across multiple processing units and GPU cores in order to reduce the latency created by the transfer of data between GPU and CPU.

REFERENCES


