Evolutionary Multi-Objective Optimization: A Distributed Computing approach

Rahul Krishna, George Mathew
Computer Science, North Carolina State University, USA
{rkrish11, george2}ncsu.edu

***Abstract*—**
***Index Terms*—**

I. Introduction

METAHEURISTIC search methods such as Evolutionary Algorithms are commonly used to *optimize* many real-world applications [1], [2]. Optimization is the task of finding solutions which satisfy one or more specified objectives. There are two types of optimizers, a single-objective optimization involves a single objective function and a single solution, a multi-objective optimization considers several objectives simultaneously. In case of a multi-objective optimizer generates a set of alternate solutions with certain trade-offs. These are called Pareto optimal solutions.

The design of the evolutionary algorithms naturally leads to parallelization. They contain several individuals which are being improved through generations. This parallel nature is particularly useful when implementing the algorithm on distributed systems. Many real-world applications have time consuming function evaluations and therefore parallelizing the evaluations on a set of available computing resources speeds up the optimization task. In addition to this, steady advances in new technologies such as Grid Computing [**?**] allows parallelization to be performed with relative ease. Several Parallel Evolutionary Algorithms have been studied in the literature (e.g., in [**?**]).

There are several parallel computing strategies (models) [3], here we explore the three main models — the Island model, Master-Slave model and Diffusion model. The current focus of this paper is the Island model, where the population for a given run is divided into semi-isolated subpopulations. Each subpopulation is assigned to a separate processor of the parallel computing system. The run begins with the one-time random creation of a separate population of individuals at each processor of the parallel computer system.

In this project, we aim to present parallel models for two evolutionary multi-objective optimizers: (1) Differential Evolution (DE) [2]; and (2) Geometric Active Learning (GALE) [4]. For implementation, we use the *henry2 Linux cluster* offered by NC State with message passing (OpenMPI) [5] a distributed programming environment.

This report is organized as follows. The following section presents a brief background on the pertinent topics. In §**??**, we discuss various strategies for parallelization. In §**??**, we provide the preliminary results. Finally, §**??** highlights how we plan on following up our work.

II. Methods and Materials

A. Evolutionary Algorithms

An Evolutionary Optimization(EO) begins its search with a population of solutions usually created at random within a specified lower and upper bound on each variable. If bounds are not supplied in an optimization problem, suitable values can be assumed only for the initialization purpose. Thereafter, the EO procedure enters into an iterative operation of updating the current population to create a new population by the use of four main operators: selection, crossover, mutation and elite-preservation. The operation stops when one or more termination criteria are met. The following evolutionary algorithms shall be parallelized.

1) Differential Evolution (DE): The Differential Evolution algorithm involves maintaining a population of candidate solutions subjected to iterations of recombination, evaluation, and selection. The recombination approach involves the creation of new candidate solution components based on the weighted difference between two randomly selected population members added to a third population member. This perturbs population members relative to the spread of the broader population. In conjunction with selection, the perturbation effect self-organizes the sampling of the problem space, bounding it to known areas of interest on the Pareto frontier. Figure 1 highlights the algorithmic details of the algorithm.

Input: $Population_{size}, Problem_{size}, Weighting_{factor}, Crossover_{rate}$
Output: S_{best}
Population $\leftarrow$ InitializePopulation($Population_{size}, Problem_{size}$)
EvaluatePopulation(Population)
$S_{best} \leftarrow$ GetBestSolution(Population)
While ($\neg$ StopCondition())
  NewPopulation $\leftarrow \emptyset$
  For ($P_i \in$ Population)
    $S_i \leftarrow$ NewSample(P_i, Population, $Problem_{size}, Weighting_{factor}, Crossover_{rate}$)
    If (Cost(S_i) $\leq$ Cost(P_i))
      NewPopulation $\leftarrow S_i$
    Else
      NewPopulation $\leftarrow P_i$
    End
  End
  Population $\leftarrow$ NewPopulation
  EvaluatePopulation(Population)
  $S_{best} \leftarrow$ GetBestSolution(Population)
End
Return (S_{best})

Fig. 1: Algorithm: Differential Evolution

2) Geometric Active Learning (GALE): GALE is a near-linear time multi-objective optimization algorithm that builds a piecewise approximation to the surface of best solutions along the Pareto frontier. GALE uses a clustering algorithm called WHERE, which is powered by a FASTMAP [6], to recursively reduce the dimensions to a single principal component. GALE offers the following advantages:

- It optimizes a problem with far fewer computations compared to other algorithms.
- It is particularly adept at handling objective functions that are non-differentiable, non-linear, multidimensional, or are subject to multiple constraints.
- It offers a very concise representation of the problem space due to WHERE.

III. Experiments

Python is our choice of programming language. This is due to its support for efficient computation frameworks like numpy [7] and scipy [8] that enables quick prototyping and benchmarking. For parallelization, we used the OpenMPI implementation of the Message Passing Interface over a python wrapper. The Open MPI Project [9] is an open source Message Passing Interface implementation that is developed and maintained by a consortium of academic, research, and industry partners.The parallelized version of the algorithm could be deployed on HPC to measure the efficiency of the algorithm. The *henry2* [10] shared memory linux cluster by NCSU may be used for this purpose. These nodes provide up to 16 shared memory processor cores and up to 128GB of memory accessible through a dedicated queue.

For our experiments, we used 4 cores using 128 GB of shared memory

A. Optimization Problem

Multi-objective Evolutionary Algorithms(MOEA) require scalable test problems that help test its efficiency. Our chosen test problem is a mathematical test problem **DTLZ2** [11], which was formulated by *Kalyanmoy* ***D****eb*, *Lothar* ***T****hiele*, *Marco* ***L****aumans* and *Eckhart* ***Z****itzler*.

Decisions : DTLZ2 has 30 decisions between 0 and 1.

$$0 \leq x_i \leq 1 \quad where \quad i = 1, 2, 3....30$$

Objectives : Although DTLZ2 allows as to generate upto n-1 objectives where n is the number of decisions, we choose to limit the number of objectives to 3 since we can model the objectives better to visualize the pareto frontier. Limiting the number of objectives also controls the domination pressure. All the three objectives needs to be minimized. The objectives are defined as follows:

$$f_1(x) = (1 + g(x_M)) \cos(x_1\pi/2)....\cos(x_{M-1}\pi/2)$$

$$f_2(x) = (1 + g(x_M)) \cos(x_1\pi/2)....\cos(x_{M-1}\pi/2)$$

$$f_3(x) = (1 + g(x_M)) \sin(x_1\pi/2)$$

$$where \quad g(x_M) = \sum_{x \in x_M} (x_i - 0.5)^2$$

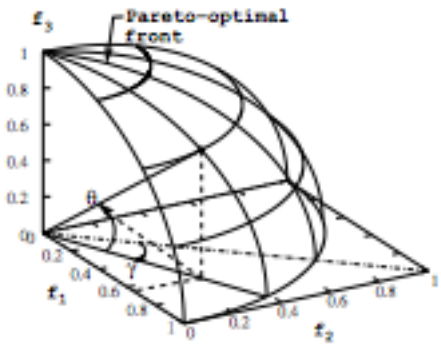

Fig. 2: Pareto Frontier of DTLZ2

Optimal Solutions : The pareto optimal solutions corresponds to the decisions $x_i = 0.5$ and all objective function values must satisfy $\sum_{m=1}^{M}(f_m)^2 = 1$. Figure 2 shows the pareto frontier that represents the optimal solutions of DTLZ2.

B. Measures

Figure 3 highlights the performance evaluation measures we use to evaluate the algorithms.

Measure	Description
Runtime	Time taken for the algorithm to be generate optimal solutions.
Convergence	Convergence is the accuracy of the obtained solutions. It mathematically represents the hypervolume of the Pareto frontier.In case of a minimization problem, we expect it to be less and large in the case of a maximization problem.
Diversity	Diversity represents the spread of the proposed solutions. Ideally the solutions should be well distributed across the Pareto frontier,rather than concentrated in certain regions.

Fig. 3: Performance Measures

C. Setup

We parallelized GALE using the "island" model [**?**]. In the "island" approach to parallelization of genetic programming, the population for a given run is divided into semi-isolated subpopulations (called demes). Each subpopulation is assigned to a separate processor of the parallel computing system. The run begins with the one-time random creation of a separate population of individuals at each processor of the parallel computer system. This is a very rudimentary approach and we will be experimenting further using a Master Slave approach.

The parameters we use for Differential Evolution and GALE are shown in 4 and 5 respectively.

Setting	Value
Population Size	100
Number of Generations	100
Mutation Rate	0.75
Crossover Probability	0.3

Fig. 4: Settings for DE

Setting	Value
Population Size	100
Number of Generations	100
Domination Factor	0.15

Fig. 5: Settings for GALE

Algorithm	Runtime(secs)	Convergence	Diversity
DE	Min : 0.45 Med : 0.49 Max : 0.81	Min : 1.80e-5 Med : 2.23e-5 Max : 2.56e-5	Min : 0.37 Med : 0.44 Max : 0.54
GALE(Serial)	Min : 39.30 Med : 42.05 Max : 43.35	Min : 5.28e-4 Med : 5.49e-4 Max : 5.73e-4	Min : 0.36 Med : 0.41 Max : 0.49
GALE(Parallel)	Min : 17.13 Med : 19.11 Max : 20.45	Min : 5.273-4 Med : 5.54e-4 Max : 5.78e-4	Min : 0.39 Med : 0.42 Max : 0.51

Fig. 6: Settings for GALE

IV. Results

The results for runtime, convergence and diversity are shown in 6. Each optimizer is run over a random initial sample of the population 20 times to get the range of output values. The minimum, median and maximum values among the 20 iterations are shown in 6. As we can see the serialized version of DE yields the lowest convergence. The diversity for all three optimizers are in the same range.

To study the distribution and rank the optimizer over all the 20 runs we use the Scott-Knott procedure recommended by Mittas & Angelis [**?**]. This method sorts a list of l treatments with ls measurements by their median score. It then splits l into sub-lists m, n in order to maximize the expected value of differences in the observed performances before and after divisions. E.g. for lists l, m, n of size ls, ms, ns where $l = m \cup n$:

$$E(\Delta) = \frac{ms}{ls} abs(m.\mu - l.\mu)^2 + \frac{ns}{ls} abs(n.\mu - l.\mu)^2$$

Scott-Knott then applies some statistical hypothesis test H to check if m, n are significantly different. If so, Scott-Knott then recurses on each division. For example, consider the following data collected under different treatments *rx*:

```
rx1 = [0.34, 0.49, 0.51, 0.6]
rx2 = [0.6,  0.7,  0.8,  0.9]
rx3 = [0.15, 0.25, 0.4,  0.35]
rx4=  [0.6,  0.7,  0.8,  0.9]
rx5=  [0.1,  0.2,  0.3,  0.4]
```

After sorting and division, Scott-Knott declares:

- Ranked #1 is rx5 with median= 0.25
- Ranked #1 is rx3 with median= 0.3
- Ranked #2 is rx1 with median= 0.5
- Ranked #3 is rx2 with median= 0.75
- Ranked #3 is rx4 with median= 0.75

Note that Scott-Knott found little difference between rx5 and rx3. Hence, they have the same rank, even though their medians differ.

Scott-Knott is better than an all-pairs hypothesis test of all methods; e.g. six treatments can be compared $(6^2 - 6)/2 = 15$ ways. A 95% confidence test run for each comparison has a very low total confidence: $0.95^{15} = 46\%$. To avoid an all-pairs comparison, Scott-Knott only calls on hypothesis tests *after* it has found splits that maximize the performance differences.

Rank	Optimizer	Median	IQR
1	DE(Serial)	2.23×10^{-5}	3.04×10^{-6}
1	GALE(Serial)	5.49×10^{-4}	8.72×10^{-6}
1	GALE(Parallel)	5.54×10^{-4}	2.21×10^{-5}

Fig. 7: Convergence of Optimizers.

Rank	Optimizer	Median	IQR
1	DE(Serial)	0.416	0.070
1	GALE(Serial)	0.431	0.056
1	GALE(Parallel)	0.432	0.047

Fig. 8: Diversity of Optimizers.

For this study, our hypothesis test H was a conjunction of the A12 effect size test of and non-parametric bootstrap sampling; i.e. our Scott-Knott divided the data if *both* bootstrapping and an effect size test agreed that the division was statistically significant (99% confidence) and not a "small" effect ($A12 \geq 0.6$).

We have estimated the Scott-Knott rankings for convergence and diversity for all three of the optimizer methods. We did not estimate it for runtime since DE was almost 50 times faster than GALE, hence there was no statistical significance to measure it.

From 7 we can see that the serialized version of DE produces the best result with rank 1, but there is no statistical difference between the serialized and parallel version of GALE in terms of convergence. The black dot in the last column shows the median value of convergence over 20 runs and the line shows the set of values between the 25th and 75th percentile. We can see that the variance for convergence is very small for all the optimizers, since the inter-quartile distance between the 25th and 75th quartile is almost 0.

The 8 shows that there is no statistical difference in diversity between all the three methods.

V. Results

References

[1] David E Goldberg. Genetic algorithm in search, optimization and machine learning, addison. *W esley Publishing Company, R eading, MA*, 1(98):9, 1989.

[2] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, December 1997.

[3] Sanaz Mostaghim, Jürgen Branke, Andrew Lewis, and Hartmut Schmeck. Parallel multi-objective optimization using master-slave model on heterogeneous resources. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1981–1987. IEEE, 2008.

[4] J. Krall, T. Menzies, and M. Davies. Gale: Geometric active learning for search-based software engineering. *Software Engineering, IEEE Transactions on*, PP(99):1–1, 2015.

[5] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[6] Christos Faloutsos and King-Ip Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *SIGMOD Rec.*, 24(2):163–174, May 1995.

[7] http://www.numpy.org/. Numpy: Fundamental package for scientific computing with python.

[8] http://www.scipy.org/. Scipy: Python-based ecosystem of open-source software for mathematics, science and engineering.

[9] http://www.open mpi.org/. Open mpi: Open source high performance computing.

[10] http://www.ncsu.edu/hpc/main.php. High performance computing: North carolina state university.

[11] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 825–830, May 2002.