Evolutionary Multi-Objective Optimization: A Distributed Computing approach

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

***Abstract*—Multi-objective problems are usually complex, NP-Hard, and resource intensive. Although exact methods can be used, they consume prohibitively large amounts of time and memory. An alternative approach would be to make use of meta-heuristic algorithms, which approximate the Pareto frontier in a reasonable amount of time. Even so, these meta-heuristic algorithms consume a significant amount of time.**

Parallel and distributed computing used in design and implementation of these algorithms may offer significant speed-ups. In addition to this, they may be used to improve the quality, increase the robustness of the the obtained solutions, and may also allow the algorithms to be scaled to solve large problems.

We present parallel models for two multi-objective optimization problems: Diffential Evolution(DE) and Geometric Active learner(GALE).

***Index Terms*—Evolutionary Algorithms, Multi Objective Optimization, Parallelization, Pareto Frontier**

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 [3] allows parallelization to be performed with relative ease. Several Parallel Evolutionary Algorithms have been studied in the literature (e.g., in [4]–[6]).

There are several parallel computing strategies (models) [7], 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) [8]. For implementation, we use the *henry2 Linux cluster* offered by NC State with message passing (OpenMPI) [9] a distributed programming environment.

This report is organized as follows. The following section presents a brief background on the pertinent topics. In §II, we discuss various strategies for parallelization. In §III, we provide the preliminary results. Finally, §V 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 two main EO we plan to study are listed below:

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.

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 recursive hierarchical clustering algorithm called WHERE, which is powered by a

```
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

Top-down Clustering with WHERE
WHERE divides data into groups of size $\alpha = \sqrt{N}$. Using this measure, WHERE runs as follows: leftmargin=3mm

1) Find two distance cases, X, Y by picking any case W at random, then setting X to its most distant case, then setting Y to the case most distant from X (which requires only $O(2N)$ comparisons).
2) Project each case Z to position x on a lines running from X to Y: if a, b are distances Z to X, Y then $x = (a^2 + c^2 - b^2)/(2ac)$.
3) Split the data at the median x value of all cases.
4) For splits larger than $\alpha = \sqrt{N}$, recurse from step1.

In terms of related work, the above is similar in approach to Boley's PDDP algorithm [11], but PDDP requires an $O(N^2)$ calculation at each recursive level to find the PCA principle component. Our method, on the other hand, performs the same task with only $O(2N)$ distance calculations using the FASTMAP heuristic [12] shown in step1. Platt [13] notes that FASTMAP is a Nyström approximation to the first component of PCA.

Fig. 2: Algorithm: WHERE

FASTMAP [10], to recursively reduce the dimensions to a single principal component, see Figure 2. After clustering, the non dominated leaf is identified(i.e the leaf with the best set of solutions) and all the points within this leaf cluster is mutated along the principal component to generate a subset of the population for the next generation. The rest of the population is randomly generated over the decision space. GALE uses only the poles(extreme points in a cluster) to check for domination. This reduces the number of evaluations drastically and improves the performance for models that takes significant time to evaluate a set of decisions.

III. Experiments

Python is our choice of programming language. This is due to its support for efficient computation frameworks like numpy [14] and scipy [15] that enables quick prototyping and benchmarking. For parallelization, we used the Open-MPI implementation of the Message Passing Interface over a python wrapper. The Open MPI Project [16] 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* [17] 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** [18], 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:

$$f_1(x) = (1 + g(x_M)) \cos(x_1\pi/2) .... \cos(x_{M-1}\pi/2)$$
$$f_2(x) = (1 + g(x_M)) \cos(x_1\pi/2) .... \cos(x_{M-1}\pi/2)$$
$$f_3(x) = (1 + g(x_M)) \sin(x_1\pi/2)$$
$$where \quad g(x_M) = \sum_{x \in x_M} (x_i - 0.5)^2$$

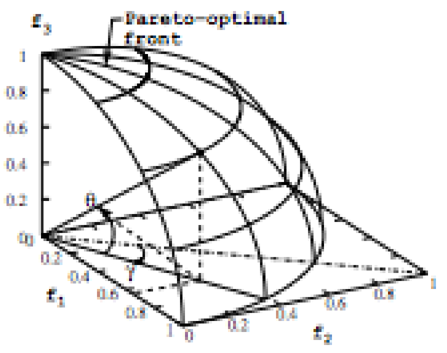

Fig. 3: Pareto Frontier of DTLZ2

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(Serial)	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
DE(Parallel)	Min : 0.28 Med : 0.30 Max : 0.31	Min : 1.80e-5 Med : 2.25e-5 Max : 2.58e-5	Min : 0.38 Med : 0.43 Max : 0.49
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: Range of Results for optimizers

Measure	Description
Runtime	Time taken for the algorithm to be generate optimal solutions.
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. 7: Performance Measures

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 3 shows the pareto frontier that represents the optimal solutions of DTLZ2.

B. Measures

Figure 7 highlights the performance evaluation measures we use to evaluate the algorithms.

C. Setup

We parallelized GALE using the ”island” model [19]. 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.

IV. Results

A. Accuracy

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 values in the 20 iterations are shown as bar charts representing the median and inter-quartile range of the 20 runs in Figure 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 provide a more comprehensive comparison, we studied the optimizers and ranked them statistically. To do this we made use of the Scott-Knott procedure recommended by Mittas & Angelis [20]. 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.

As a example, consider the following hypothetical 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.

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

Processors	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
GALE	96.38	50.32	34.96	25.92	21.46	17.98	14.95	13.17	12.35	10.56	10.13	9.14	8.43	7.67	7.58	6.97
DE	0.97	0.53	0.3	0.23	0.17	0.15	0.13	0.11	0.11	0.08	0.09	0.14	0.1	0.09	0.07	0.06

TABLE I: Runtimes of GALE and DE in seconds

Rank	Optimizer	Median	IQR	
1	DE(Parallel)	2.30×10^{-5}	2.74×10^{-6}	●
1	DE(Serial)	2.36×10^{-5}	3.04×10^{-6}	●
2	GALE(Serial)	5.49×10^{-4}	8.72×10^{-6}	●
2	GALE(Parallel)	5.54×10^{-4}	2.21×10^{-5}	—●—

Fig. 8: Convergence of Optimizers.

Rank	Optimizer	Median	IQR	
1	GALE(Parallel)	0.416	0.070	●—
1	DE(Parallel)	0.417	0.049	●
1	DE(Serial)	0.431	0.056	●-
1	GALE(Serial)	0.432	0.047	●

Fig. 9: Diversity of Optimizers.

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 figure 8 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 figure 9 shows that there is no statistical difference in diversity between all the three methods.

B. Runtimes

HPC supports upto 16 processors with 128GB of shared memory. The runtimes for DE and GALE when deployed on 1 to 16 processors is tabulated in table I. The results show that after 8 processors the runtime for DE starts to become constant and for GALE, the results does not stabilize. This might be because the runtimes for GALE is two orders of magnitude more than that of DE and hence the communication overhead might not be significant in its case.

Based on the data in table I, speed up for on different processors is illustrated in 10. Speed up is calculated as

$$SpeedUp = \frac{T_{Serial}}{T_{Parallel}}$$

where T_{Serial} is the time taken to run the optimizer on a single processor and T_n is the time taken to run the optimizer parallely on **n** number of processors where n varies from 2 to 16.

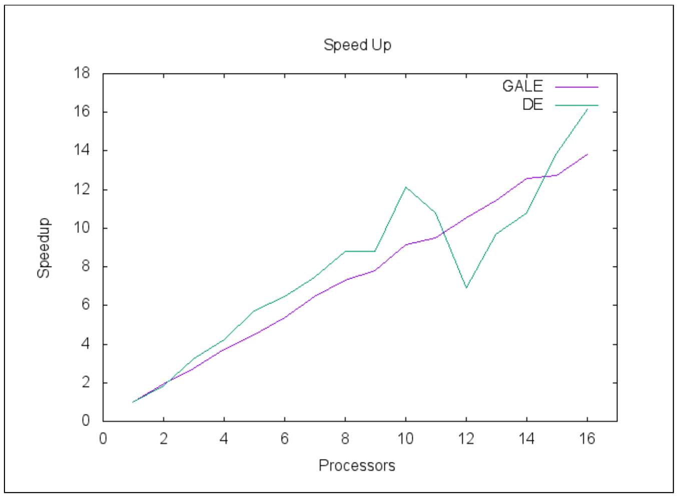

Fig. 10: Speed Ups for GALE and DE

V. Future Work

Now that we have implemented the serialized version of the algorithms and a parallelized version of GALE, we plan to:

1) Experiment different parallelization strategies for GALE
2) Check the scalability of the optimizer on a larger model like XOMO [21]

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 &ndash; a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, December 1997.

[3] David Abramson, Andrew Lewis, and Tom Peachey. Nimrod/o: A tool for automatic design optimization. 2000.

[4] Enrique Alba and Marco Tomassini. Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5):443–462, 2002.

[5] Jürgen Branke, Andreas Kamper, and Hartmut Schmeck. Distribution of evolutionary algorithms in heterogeneous networks. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 923–934. Springer, 2004.

[6] Kalyanmoy Deb, Pawan Zope, and Abhishek Jain. Distributed computing of pareto-optimal solutions with evolutionary algorithms. In *Evolutionary Multi-criterion optimization*, pages 534–549. Springer, 2003.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Daniel Boley. Principal direction divisive partitioning. *Data mining and knowledge discovery*, 2(4):325–344, 1998.

[12] Christos Faloutsos and King-Ip Lin. FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, 1995.

[13] John C. Platt. FastMap, MetricMap, and Landmark MDS are all Nyström algorithms. In *In Proceedings of 10th International Workshop on Artificial Intelligence and Statistics*, pages 261–268, 2005.

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

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

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

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

[18] 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.

[19] Steven Gustafson and Edmund K Burke. The speciating island model: An alternative parallel evolutionary algorithm. *Journal of Parallel and Distributed Computing*, 66(8):1025–1036, 2006.

[20] Nikolaos Mittas and Lefteris Angelis. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. *IEEE Trans. Software Eng.*, 39(4):537–551, 2013.

[21] Tim Menzies and Julian Richardson. Xomo: Understanding development options for autonomy. In *COCOMO forum*, volume 2005, 2005.