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 §VI, we provide our experimental results. §III we discuss the problems we are studying followed by the parallelization strategies in §V. In §VI and §VII we describe our experiments and discuss the results. §VIII summarizes the conclusions we can draw from the experiments. Finally, §IX highlights our work can be enhanced.

II. 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:

A. 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 ← InitializePopulation(Population_size, Problem_size)
EvaluatePopulation(Population)
S_best ← GetBestSolution(Population)
While ( ¬ StopCondition())
  NewPopulation ← ∅
  For (P_i ∈ Population)
    S_i ← NewSample(P_i, Population, Problem_size, Weighting_factor, Crossover_rate)
    If (Cost(S_i) ≤ Cost(P_i))
      NewPopulation ← S_i
    Else
      NewPopulation ← P_i
    End
  End
  Population ← NewPopulation
  EvaluatePopulation(Population)
  S_best ← GetBestSolution(Population)
End
Return (S_best)
```

Fig. 1: Algorithm: Differential Evolution

B. 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 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. Optimization Problem

Multi-objective Evolutionary Algorithms(MOEA) require scalable test problems that help test its efficiency.

A. DTLZ2

DTLZ2 is a mathematical test problem [14], 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:

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

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

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

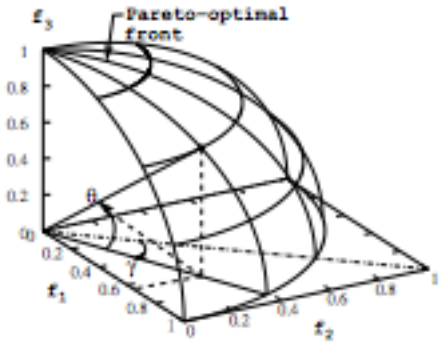

Fig. 3: Pareto Frontier of DTLZ2

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

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

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. POM3

POM3 is a simulation of requirement prioritization strategies proposed by **P**ort, **O**klov and **M**enzies [15]. POM3 has 9 decisions and its possible values are described in table I. POM3 has 4 objectives.

- **Cost :** Cost of completing the project. To be ***minimized***.

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	Convergence	Diversity
DE(Serial)	Min : 1.80e-5 Med : 2.23e-5 Max : 2.56e-5	Min : 0.37 Med : 0.44 Max : 0.54
DE(Parallel)	Min : 1.80e-5 Med : 2.25e-5 Max : 2.58e-5	Min : 0.38 Med : 0.43 Max : 0.49
GALE(Serial)	Min : 5.28e-4 Med : 5.49e-4 Max : 5.73e-4	Min : 0.36 Med : 0.41 Max : 0.49
GALE(Parallel)	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

- **Utility :** Normalized Utility of the project. To be ***maximized***.
- **Completion Percentage :** Expected percentage of project that can be completed. To be ***maximized***.
- **Idle Time :** Total unproductive time of developers. To be ***minimized***.

C. XOMO

XOMO is a Monte Carlo Simulator which models NASA's space program software [16]. XOMO has 23 decisions and are described in figure 7. XOMO has 4 objectives all of which have to be minimized.

- **Effort :** Total effort required by the developer to complete the project.
- **Months :** Total number of months required to complete the project.
- **Defects :** Total number of "bugs" in the project.
- **Risk :** Risk involved in completing the project.

D. ERS

Emergency **R**esponse **S**ystem (ERS) is a software feature model[1]. A feature model is a compact representation of all the products of the Software Product Line in terms of "features". Feature models are visually represented by means of feature diagrams. Parent and Child features are categorized as

- **Mandatory :** Child feature is required.
- **Optional :** Child feature is optional.
- **Or :** Atleast one of the sub-features must be selected.

[1]https://en.wikipedia.org/wiki/Feature_model

Decision	Range
Culture	$0.10 < x < 0.90$
Criticality	$0.82 < x < 1.26$
Criticality Modifier	$0.02 < x < 0.10$
Initial Known	$0.40 < x < 0.70$
Inter-Dependency	$0.0 < x < 1.0$
Dynamism	$1.0 < x < 50.0$
Size	$x \in [3, 10, 30, 100, 300]$
Team Size	$1.0 < x < 44.0$
Plan	$0 < x < 4$

TABLE I: Decisions for POM3

- **Alternative :** One of the sub-features must be selected.

ERS contains 35 decisions and 3 objectives.

IV. Measures

The accuracy and effectiveness of the parallelization strategies are indicated by some mathematical measures. We use 4 such evaluation metrics in our experiments and it is highlighted in Figure 8.

Measure	Description
Runtime	Time taken for the algorithm to be generate optimal solutions.
Speed Up	Ratio of time taken for the parallelized version of the algorithm over the time taken for the serial version.
Convergence	Convergence represents the accuracy of the obtained solutions. It is the distance between the obtained solutions and ideal Pareto frontier.
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. 8: Performance Measures

V. Parallelization Strategies

A. Island Model

In the "island" approach to parallelization of genetic programming [17], 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. Each processor runs an instance of the optimizer on the sub-population and the final sub-population across each processor is aggregated and represents the approximate Pareto Frontier. This algorithm takes advantage of the randomness of the optimizer in mutation and achieves similar results to optimizing a population on a single processor.

	Definition	Low-end	Medium	High-end
Scale factors:				
flex	development flexibility	development process rigorously defined	some guidelines, which can be relaxed	only general goals defined
pmat	process maturity	CMM level 1	CMM level 3	CMM level 5
prec	precedentedness	we have never built this kind of software before	somewhat new	thoroughly familiar
resl	architecture or risk resolution	few interfaces defined or few risk eliminated	most interfaces defined or most risks eliminated	all interfaces defined or all risks eliminated
team	team cohesion	very difficult interactions	basically co-operative	seamless interactions
Effort multipliers				
acap	analyst capability	worst 15%	55%	best 10%
aexp	applications experience	2 months	1 year	6 years
cplx	product complexity	e.g. simple read/write statements	e.g. use of simple interface widgets	e.g. performance-critical embedded systems
data	database size (DB bytes/SLOC)	10	100	1000
docu	documentation	many life-cycle phases not documented		extensive reporting for each life-cycle phase
ltex	language and tool-set experience	2 months	1 year	6 years
pcap	programmer capability	worst 15%	55%	best 10%
pcon	personnel continuity (% turnover per year)	48%	12%	3%
plex	platform experience	2 months	1 year	6 years
pvol	platform volatility ($\frac{frequency\ of\ major\ changes}{frequency\ of\ minor\ changes}$)	$\frac{12\ months}{1\ month}$	$\frac{6\ months}{2\ weeks}$	$\frac{2\ weeks}{2\ days}$
rely	required reliability	errors mean slight inconvenience	errors are easily recoverable	errors can risk human life
ruse	required reuse	none	multiple program	multiple product lines
sced	dictated development schedule	deadlines moved closer to 75% of the original estimate	no change	deadlines moved back to 160% of original estimate
site	multi-site development	some contact: phone, mail	some email	interactive multi-media
stor	main storage constraints (% of available RAM)	N/A	50%	95%
time	execution time constraints (% of available CPU)	N/A	50%	95%
tool	use of software tools	edit,code,debug		integrated with life cycle

Fig. 7: Decisions for XOMO

B. Master Slave Model

The Master slave model of parallelization uses one processes as a master and all the remaining processors as slaves for optimization [18]. For each generation of optimization, the master selects a sub-population for each free slave processor. The size of the sub-population is equal to size of the population divided by the number of slaves. The slave evaluates the fitness and computes the best solution(s) for each population set. The slave then mutates the sub-population and sends the mutants and the evaluations back to the master. The master aggregates results from all the slaves and repeats until the number of generations are completed.

VI. Experimental Setup

Python is our choice of programming language. This is due to its support for efficient computation frameworks like numpy[2] and scipy[3] that enables quick prototyping and benchmarking. For parallelization, we used the OpenMPI implementation of the Message Passing Interface over a python wrapper and the "multiprocessing"[4] package of python. The Open MPI Project[5] 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

[2]http://www.numpy.org
[3]http://www.scipy.org
[4]https://docs.python.org/2/library/multiprocessing.html
[5]http://www.openmpi.org

the efficiency of the algorithm. The *henry2*[6] 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 to evaluate the convergence and diversity, we used 4 cores using 128 GB of shared memory. To evaluate the runtimes and speedup, the experiments were conducted on 1 to 16 cores with 128 GB of shared memory. The parameters we use for Differential Evolution and GALE are shown in 4 and 5 respectively.

VII. 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 [19]. 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.

[6]https://ncsu.edu/hpc/

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. 9: 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. 10: 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 figure 9 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 10 shows that there is no statistical difference in diversity between all the three methods.

B. Performance

HPC supports upto 16 processors with 128GB of shared memory. Speed up of a processor 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.

STRATEGY	MODEL	1	2	4	6	8	10	12	14	16
Island	**GALE**	96.38	50.32	25.92	17.98	13.17	10.56	9.14	7.67	6.97
Island	**DE**	0.97	0.53	0.23	0.15	0.11	0.08	0.14	0.09	0.06
Master-Slave	**GALE**	68.3	18.16	4.92	2.38	1.59	1.25	0.99	0.91	0.86
Master-Slave	**DE**	4.05	1.43	0.61	0.31	0.22	0.18	0.14	0.12	0.14

Fig. 11: Runtimes of DTLZ-2 using GALE and DE in seconds

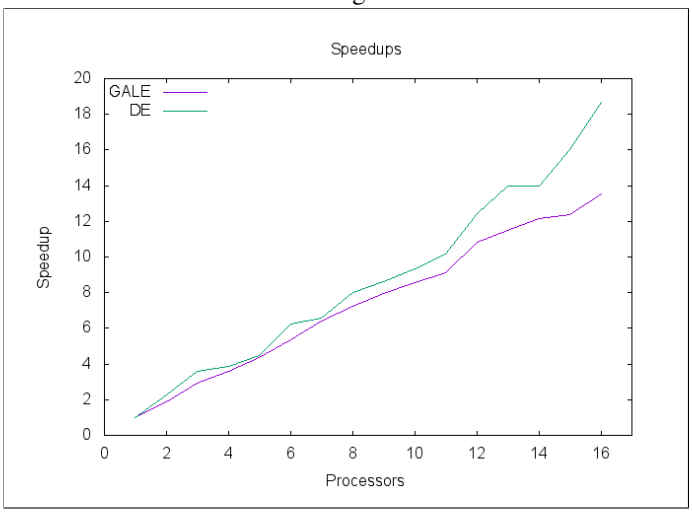

Fig. 12: Island Model DTLZ2 Speed Ups

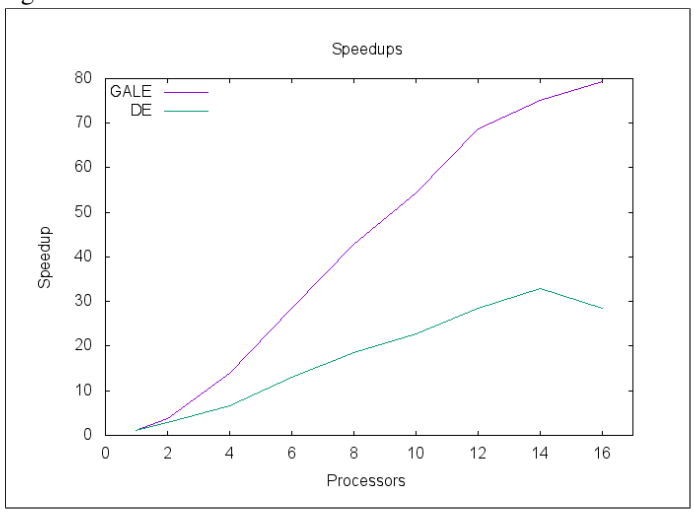

Fig. 13: Master Slave Model DTLZ2 Speed Ups

STRATEGY	MODEL	1	2	4	6	8	10	12	14	16
Island	**GALE**	229.98	127.21	65.51	44.92	36.93	28.19	24.93	21.64	22.37
Island	**DE**	990.81	790.28	479.78	375.42	294.41	244.17	188.09	163.16	145.67
Master-Slave	**GALE**	29.97	28.56	43.24	31.91	27.59	31.58	38.62	27.76	48.45
Master-Slave	**DE**	265.15	130.48	127.45	75.31	47.91	51.74	51.11	58.64	35.60

Fig. 14: Runtimes of POM3 using GALE and DE in seconds

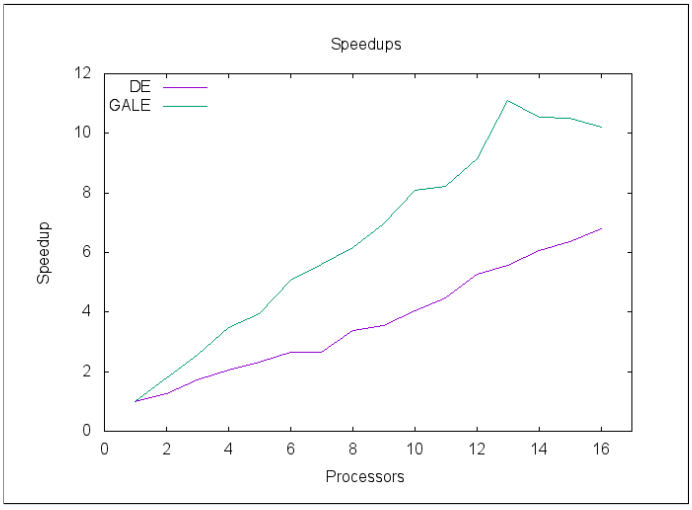

Fig. 15: Island Model POM3 Speed Ups

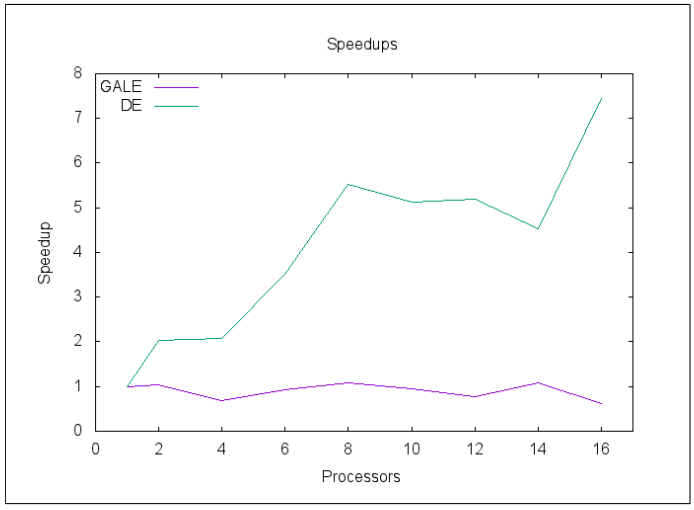

Fig. 16: Master Slave Model POM3 Speed Ups

STRATEGY	MODEL	1	2	4	6	8	10	12	14	16
Island	**GALE**	229.09	112.21	59.90	41.07	30.97	26.25	22.37	19.01	16.56
Island	**DE**	8.53	4.48	2.33	1.61	1.17	0.96	0.77	0.67	0.62
Master-Slave	**GALE**	86.49	33.67	18.16	14.94	13.09	12.04	11.69	11.80	10.56
Master-Slave	**DE**	60.93	40.12	20.71	14.34	10.22	8.21	7.46	6.27	5.14

Fig. 17: Runtimes of XOMO using GALE and DE in seconds

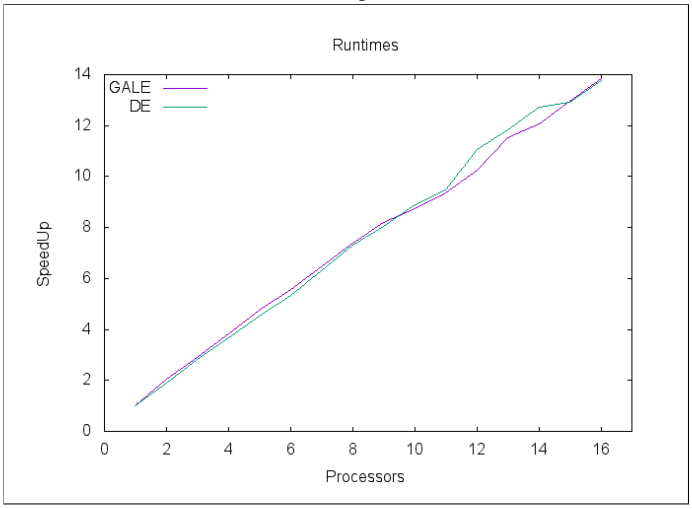

Fig. 18: Island Model XOMO Speed Ups

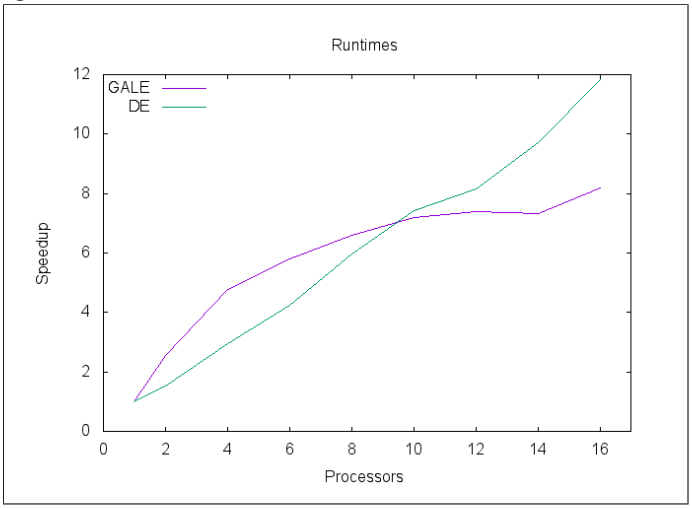

Fig. 19: Master Slave Model XOMO Speed Ups

STRATEGY	MODEL	1	2	4	6	8	10	12	14	16
Island	**GALE**	138.19	65.22	43.46	29.26	21.84	18.29	16.24	14.47	13.97
Island	**DE**	204.61	103.17	53.05	34.65	26.98	29.25	29.29	28.64	29.38

Fig. 20: Runtimes of ERS using GALE and DE in seconds

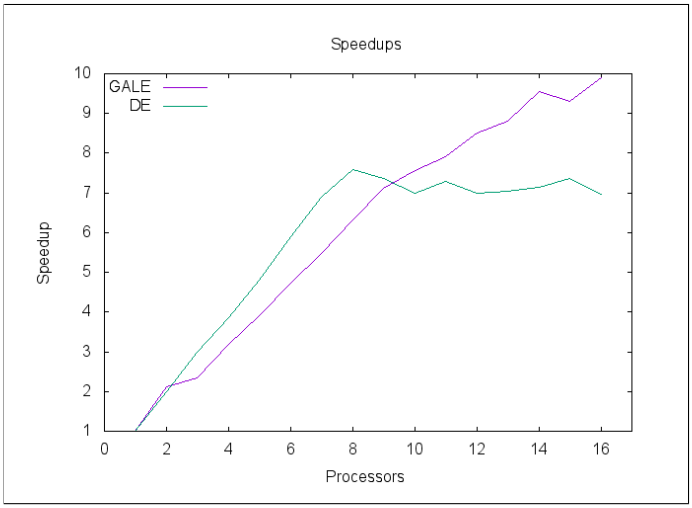

Fig. 21: Island Model ERS Speed Ups

1) DTLZ2: The runtimes for DE and GALE on both the island and master-slave model when deployed on 1 to 16 processors is tabulated in figure 11 and speedups are shown in figures 12 and 13. The results show that there is a steady increase in speedup for island model. Although, for the master slave model, GALE has significant increase in speedups, there is a plateauing effect observed for DE.

2) POM3: The runtimes for DE and GALE on both the island and master-slave model when deployed on 1 to 16 processors is tabulated in figure 14 and speedups are shown in figures 15 and 16. POM3 is computationally expensive and as we can see GALE outperforms DE in both the master slave and island model. In the master slave model, we observe that increasing the number of processors does not aid in better runtimes. This can be drilled down to the high communication overhead due to frequent communications in master slave over the island models.

3) XOMO: The runtimes for DE and GALE on both the island and master-slave model when deployed on 1 to 16 processors is tabulated in figure 17 and speedups are shown in figures 18 and 19. XOMO is fast for a real world model and we can see that DE performs better than GALE. In the island model we observe almost exact speedups in both DE and GALE. In the master slave model we can see a plateauing effect for GALE.

4) ERS: ERS was implemented only on the island model. The runtimes for DE and GALE when deployed between 1 to 16 processors is shown in Figure 20 and the speedups are shown in 21. We can see that GALE is faster than DE for this model due to the large evaluation time for GALE.

VIII. Conclusion

Based on our experiments we draw following conclusions:

- Parallelization strategies are effective for Multi Objective Optimization problems.
- For mathematical models or real world models which are not compute intensive, we suggest using Differential Evolution.
- For models, which require a lot of time for evaluation, we propose GALE would be optimal.

IX. Future Work

This project cannot be deemed as complete but just a study on the parallelization capabilities of an MOEA. The following enhancements can be performed in the immediate future:

1) Explore mutation strategies for heavily constrained real world problems.
2) Study these parallelization strategies for other MOEAs like NSGA2, SPEA etc.
3) While parallelization if we divide the feature space more effectively based on the problem space, we can achieve faster convergence to the Pareto Front. This is another avenue that can be explored.

X. Acknowledgement

This project is part of the course CSC 491/591 - Experimental Algorithms. We would like to thank the course staff Dr Matthias Stallmann for giving us the oppurtunity to work on this project and his constant support. We also express our gratitude to NCSU for providing us with the resources to run our experiments.

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] 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] 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.

[15] Boehm and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

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

[17] 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.

[18] Alexandru-Ciprian Zvoianu, Edwin Lughofer, Werner Koppelsttter, Gnther Weidenholzer, Wolfgang Amrhein, and ErichPeter Klement. On the performance of master-slave parallelization methods for multi-objective evolutionary algorithms. In Leszek Rutkowski, Marcin Korytkowski, Rafa Scherer, Ryszard Tadeusiewicz, LotfiA. Zadeh, and JacekM. Zurada, editors, *Artificial Intelligence and Soft Computing*, volume 7895 of *Lecture Notes in Computer Science*, pages 122–134. Springer Berlin Heidelberg, 2013.

[19] 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.