https://doi.org/10.48047/AFJBS.6.14.2024.6109-6121

African Journal of Biological Sciences

Journal homepage: http://www.afjbs.com

Research Paper Open Access

An Efficient Population based Parallel Search Approach to Optimize Equations for High Dimensional Space

Vedatrayee Chatterjee[1], Dr. Kamal Dhanda[2]

[1]*Research Scholar, Department of Computer Science and Engineering, School of Engineering & Technology,Om Sterling Global University,Hisar 125001, INDIA.*

[2]*Associate Professor, Department of Computer Science and Engineering, School of Engineering & Technology,Om Sterling Global University,Hisar 125001, INDIA.*

Volume 6, Issue 14, Aug 2024

Received: 15 June 2024

Accepted: 25 July 2024

Published: 15 Aug 2024

doi: 10.48047/AFJBS.6.14.2024.6109-6121

Abstract

The utilization of parallel search techniques is increasingly recognized for its efficacy in modeling, simulating, and comprehending intricate real-world phenomena. This paper explores the application of parallel search techniques to address Prostate Cancer classification. Emphasis is placed on ensuring that each point within the search space is traversed only once, leading to the acquisition of optimized values. The methodology entails employing a straightforward operation to explore points within a search space of 10^n dimensions. The proposed technique efficiently navigates through every search point precisely once, ensuring comprehensive coverage of the search space. The selection of search points throughout the entirety of the search space is facilitated by a method termed RLC.

Keywords: Parallel search, Rotate Left and Complement (RLC) operator, Search space, Normal Generators, Exceptional Generators, Optimization

1. Introduction

Optimization is the process of finding the greatest or least value of a function for some constraint, which must be true regardless of the solution. In other words, optimization finds the most suitable value for a function within a given domain. In mathematical terms, an optimization problem is the problem of finding the best solution from among the set of all possible solutions. In other words, optimization means finding an alternative with the most cost effective or highest achievable performance under the given constraints, by maximizing desired factors and minimizing undesired ones. The objective is to find the solution to a predefined objective function via an iterative process, towards an optimal value. In optimization problems, a mathematical representation of the objective function is clearly defined along with its constraints [1].

In an optimization problem, the solution technique or algorithms used for optimization is determined by the types of mathematical relationships between the objective and constraints and the decision variables.

Categorizing the optimization model is a significant stride in the optimization process, as algorithms for solving optimization problems are bespoke to a particular type of problem.

If the variables can take any value in within a given range, then it is called continuous optimization. Continuous optimization can further be categorized as unconstrained optimization, where there is no constraint on the variables, and constrained optimization, where there are constraints on the variables. In discrete optimization, some or all the variables can take a value from a discrete set of values.

In this paper we introduce a new, efficient, distributed as well as parallel computing strategy for exponential (10^n) search space with the above objectives. The results shows that the proposed technique is really efficient in terms of time and suitable for highly distributed and parallel environment and is found to provide better results. But describing the proposed technique, some we have discussed some important points.

The organization of this article is as follows, Section 2 depicts problem with existing methods. Section 3 describes the proposed technique. Some Benchmark functions are used for optimization using proposed technique and comparative studies have been observed in section 4. Section 5 contains the conclusion.

2. Problems with existing methods

While there exists a plethora of algorithms in the literature suitable for solving optimization problems, none of them offer a universal solution applicable to all scenarios [2]. Certain algorithms may excel in specific problem domains while faltering in others [3] [5].

One such algorithm, the Genetic Algorithm (GA) [9], leverages principles from natural genetics to facilitate function optimization. Despite its utility, GA is not without its drawbacks. Notably, GA relies on numerous tunable parameters, and while some guidelines exist for parameter selection, there is no definitive method for choosing these values. Consequently, executing GA for the same problem can yield vastly different outcomes with the same number of iterations or generations. Although GA [12] is renowned for providing optimal solutions as iterations approach infinity, practical constraints necessitate terminating the process after a finite number of iterations [4]. This limitation compels users to fine-tune parameter values. GA initiates with a population of randomly generated solutions, with subsequent iterations generating a new population through selection, crossover, and mutation operations [8]. Ultimately, this process yields a population of solutions.

3. Proposed Technique

In this paper we have proposed a technique to optimize some benchmark functions for exponential (10^n) search space. So the optimal value obtained is always a decimal integer. The proposed algorithm is fast, and suitable for highly distributed and parallel environment.

The algorithm starts with an initial population of distinct solutions. As the algorithm proceeds, we always get a distinct population of solutions at each generation. The process is repeated until the whole search space is covered. No point appeared more than once which saves the execution time. Besides, it always finds distinct points in the search space and hence increases the possibility of getting good results.

In the proposed searching technique the entire search space is split in to several subsets. The number of subset is a function of the length of the search space. For example if the length of the string to be searched is three, then the size of the search space is 10^3=1000. This search space is divided in to 170 subsets. Among them, 165 subsets contain 6 elements each and rest 5 subsets contain 2 elements each. The technique of dividing the search space in to subsets has been explained later. So the searching of the element can be performed from 170 different search points and the process of searching the element from those 170 search points can be performed in parallel.

The initial search points start from ***n*** number of 0's. That is, if the size of the search space is 10^3, then the starting search point (which has been termed as Generator later) is 000. Then the other initial search points are obtained using some algorithm.

The most important issues are how to get the starting search points and what operators are to be used to get the distinct solutions. In this section we will describe an illustrative example that will help us to understand the whole investigative work throughout the paper.

The notations used are as below:

$\mathbf{S_i}$ = String of length n
D = Total search Space = $\mathbf{10^n}$
G = Total number of Normal Generators
E = Total number of Exceptional Generators
$\boldsymbol{\alpha_k}$ = Maximum value of normal generator for **k** digit decimal string

A decimal coded string of length **n** is used as a representation of each search point. Each string **S** in the search space **D** is of the form $\mathbf{S}= (\mathbf{d_1}, \mathbf{d_2}\ldots \mathbf{d_n})$, where $\mathbf{d_i} \in \{0, 1, 2\ldots9\}, \forall\ \mathbf{i}$. The total number of strings in decimal representation is $\mathbf{10^n}$. We propose to find the optimal string(s) among these strings. A finite and distinct sample of initial solutions, each of length **n**, is drawn from $\mathbf{D\ (10^n)}$ to form the initial population **P**. To incorporate variation within the solutions, a **R**otate **L**eft and **C**omplement operator (**RLC**) has been used [9] [13]. The **RLC** operator is described below:

Suppose we have a solution $\mathbf{s_i}$ of length is 5 (n = 5) at instance t. Let it be $\mathbf{s_i}$ = 02341. Using the RLC operator, it is possible to generate $\mathbf{2 \times n = 2 \times 5 = 10}$ different solutions (including $\boldsymbol{s_i}$) from a single solution $\boldsymbol{s_i}$.

The string **0**<u>2341</u> produces <u>2341</u>**9** using RLC operator**.** The underlined portion of the string is shifted 1 position to the left and 9's complement of the left most digit of the old string is placed at the unit position of the new string. Thus the generated strings are 23419, 34197, 41976, 19765, 97658, 76580, 65802, 58023, 80234 over (**2*n-1**) iterations. The process of generating strings is stopped when the initial string comes back. Thus from a search string of length n, we can obtain (2*n-1) new distinct search strings using the RLC operator. The string 02341 is called a ***Normal Generator***, as it can generate (2*n-1) number of distinct strings. If RLC operator is applied on a string generated from the normal generator, then the same strings are generated which have already been generated from the normal generator. For any particular value of ***n***, there are a fixed number of normal generators. A string is said to be an ***Exceptional Generator*** if it does not produce (2*n − 1) different strings by successive application of RLC operator. Table 1 depicts an example of normal and exceptional generators for n=1, 2, 3.

Table 1
Normal and Exceptional Generators

String Length	**Values of Normal Generator**	**Values of Exceptional Generator**
1	0,1,2,3.4	Nil
2	0,1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,11 ,12 ,13 ,14 ,15 ,16 ,17 ,22 ,23 ,24 ,25 ,26 ,33 ,34 ,35 ,44	Nil
3	0, 1, 2, 3, 4, 5 ,6 ,7 ,8 ,10 ,11 ,12 ,13 ,14 ,15 ,16 ,17 ,18 ,20 ,21 ,22 ,23 ,24 ,25 ,26 ,27 ,28 ,30 ,31 ,32 ,33 ,34 ,35 ,36 ,37 ,38 ,40 ,41 ,42 ,43 ,44 ,45 ,46 ,47 ,48 ,50 ,51 ,52 ,53 ,54 ,55 ,56 ,57 ,58 ,60 ,61 ,62 ,63 ,64 ,65 ,66 ,67 ,68 ,70 ,71 ,72 ,73 ,74 ,75 ,76 ,77	090, 181, 272, 363, 454

	,78 ,80 ,81 ,82 ,83 ,84 ,85 ,86 ,87 ,88 ,111 ,112 ,113 ,114 ,115 ,116 ,117 ,121 ,122 ,123 ,124 ,125 ,126 ,127 ,131 ,132 ,133 ,134 ,135 ,136 ,137 ,141 ,142 ,143 ,144 ,145 ,146 ,147 ,151 ,152 ,153 ,154 ,155 ,156 ,157 ,161 ,162 ,163 ,164 ,165 ,166 ,167 ,171 ,172 ,173 ,174 ,175 ,176 ,177 ,222 ,223 ,224 ,225 ,226 ,232 ,233 ,234 ,235 ,236 ,242 ,243 ,244 ,245 ,246 ,252 ,253 ,254 ,255 ,256 ,262 ,263 ,264 ,265 ,266 ,333 ,334 ,335 ,343 ,344 ,345 ,353 ,354, 355, 444	

There are 1000 points in the search space for string length 3. Here we find that only 170 (165 normal and 5 exceptional) generators are enough to traverse the whole 10^3 = 1000 search points in parallel. Note that no point is common between any two sets of points generated by any two of the above 170 generators.

Table 2, depicts the string length (***n***) and corresponding number of normal generators and exceptional generators.

Table-2

Number of Normal and Exceptional Generators

String Length	**No. of Normal Generator**	**No. of Exceptional Generator**
1	5	0
2	25	0
3	165	5
4	1250	0
5	9999	5
6	83325	25
7	714285	5
8	6250000	0
9	555555500	170

From the table we find that as the string length is increased, the number of generators is also increasing almost exponentially.

Table 3 provides the number of normal generators needed for searching the space. It is self -explanatory. Total search space = number of normal generators × number of iteration for normal generator + number. of exceptional normal generators × no. of points to be searched by exceptional generators. For example, for a string of length 5, we need 10004 generators (9999 normal and 5 exceptional) to traverse 10^5 = 100000 points in parallel.

Table 3

Number of generators for string length *n*

String Length (n)	**Size of Search Space Base^n= x*a + y*b**	**No. of Normal Generator (x)**	**No of Iteration for Normal Generator (a)**	**No. of Exceptional Generator (y)**	**No of Iteration for Exceptional Generator (b)**
1	10^1 = 5*2 + 0	5	2	0	0
2	10^2 = 25*4 + 0	25	4	0	0
3	10^3 = (165*6) + (5*2)	165	6	5	2
4	10^4 = (1250*8) + (0*0)	1250	8	0	0
5	10^5 = (9999*10) + (5*2)	9999	10	5	2

6	10^6 = (83325*12) + (25*4)	83325	12	25		4
7	10^7 = (714285*14) + (5*2)	714285	14	5		2
8	10^8 = (6250000*16) + (0*0)	6250000	16	0		0
9	10^9 = (555555500*18) + (5*2)+(165*6)	555555500	18	170	165	6
					5	2

The maximum value of Normal Generator for string of length n can be represented as a function of string length (n) and the maximum value of Normal Generator for string of length *n-1*.

Let the maximum value of normal generator is α_n where ***n*** is the length of the string. The minimum string length is 1 and it has been extended up to **10**. The maximum value normal generator is calculated according to the following formula:

- $\alpha_n = 10^* \alpha_{n-1} + 4$ When n>2 and n is an Odd number
- $\alpha_n = 10^* (\alpha_{n-1} +1) + 4$ When n>2 and n is an Even number

For example:

- $\alpha_0 = 0$
- $\alpha_1 = 10^* \alpha_0 + 4 = 10^*0 + 4 = 4$
- $\alpha_2 = 10^* \alpha_1 + 4 = 10^*4 + 4 = 44$
- $\alpha_3 = 10^* \alpha_2 + 4 = 10^*44 + 4 = 444$
- $\alpha_4 = 10^* (\alpha_3 +1) +4 = 10^*(444 + 1) + 4 = 4454$ n>3 and n is an even number
- $\alpha_5 = 10^* \alpha_4 +4 = 10^*4454 + 4 = 44544$
- $\alpha_6 = 10^* (\alpha_5 +1) +4 = 10^*(44544 + 1) + 4 = 445454$ n>3 and n is an even number

Table 4 enlists the string length and the corresponding maximum value of normal generator.

Table 4
Maximum value of Normal Generator

String Length	**Maximum value of Normal Generator**
1	4
2	44
3	444
4	4454
5	44544
6	445454
7	4454544
8	44545454
9	445454544

The feature to be noted that, except the maximum value of normal generator of string length 4, 6 and 8; all other maximum value of normal generators are palindrome.

It has already been mentioned that a string is said to be an ***Exceptional Generator*** if it does not produce (***2*n − 1***) different strings by successive application of RLC operator. For different string length, number of

exceptional generator is also different. The following features of exceptional generators have been exhaustively examined:

i) For string length 2^i, i=0, 1, 2, 3… number of exceptional generator is 0.

ii) If the string length is an odd number (string length ≥2 and string length ≤7), then there is always five (5) numbers of exceptional generators. From the observation, the formula to find out the t^{th} exceptional generator of string length n is given below:

$$\mathbf{t^{th}} \text{ exceptional generator} = \mathbf{t} \times \sum_{\mathbf{i=2,4,6,..}}^{\mathbf{n-i}} \left(\mathbf{9} \times \mathbf{10^{n-i}} + (\mathbf{t-1})\right)$$

Where-$\boldsymbol{t \geq 1}$ and $\boldsymbol{t \leq 5}$ and $\boldsymbol{n - i \geq 1}$

iii) There is a very interesting point to be noted that, all the above Exceptional Generators are palindrome number.

iv) There is a relation between two consecutive exceptional generators:

- The common difference between two consecutive Exceptional Generators of string length 3 is 91.
- The common difference between two consecutive Exceptional Generators of string length 5 is 9091.
- The common difference between two consecutive Exceptional Generators of string length 7 is 909091.

v) There are 170 Exceptional Generators of length 9. Among them, 165 have 6 iterations and rest 5 has 2 iterations.

Table 5 contains the exceptional generators of string length 3, 5 and 7:

Table 5
Exceptional Generators

String Length	Exceptional Generators
3	090, 181, 272, 363, 454
5	09090, 18181, 27272, 36363, 45454
7	0909090, 1818181, 2727272, 3636363, 4545454

Proposed exhaustive parallel search algorithm

We begin with P, generator starting from 0 and until it reaches up to the maximum value, i.e.α_n. Each generator is allowed to repeat for 2n iteration.

We have stored all the populations in a file (FP). We then increment the counter by 1 and check whether it is present in file or not. If present then increment by 1 and so on.

The Proposed exhaustive parallel search algorithm:

1. For length n, i=0 to Max_Value (α_n), (for normal generators) increment by 1(checked in stored population file FP) excluding the numbers whose unit value is 9.
2. Repeat for 2n times by applying RLC operator on P.
3. Repeat step 2 for exceptional generators.
4. Print results.

We have used one processor for the above algorithm. For the purpose of parallel implementation, we would like to use at least two processors to reduce the computational time. Our algorithm is divided into two independent modules that can be easily executed in parallel.

Processor 1:

1. For i=0 to m (for normal generators) increment by 1(checked in stored population file FP1) excluding the numbers whose unit value is 9.
2. Repeat for 2*n* times by applying RLC operator on P.
3. Repeat step 2 for exceptional generators.
4. Print results.

Processor 2:

1. For *i=m+1* to Max_Value (α_n) (for normal generators) increment by 1(checked in stored population file FP2) excluding the numbers whose unit value is 9.
2. Repeat for 2*n* times by applying RLC operator on P.
3. Repeat step 2 for exceptional generators.
4. Print results.

Remarks:

1. Here m is the boundary between the processors for distributing the work load among them. Both the processors may function independent of each other.
2. For p>2, it can be suitably changed the value of m for parallel implementation.

4. Results and comparative studies

The above stated technique has been used to optimize a set of equations or benchmark functions [10] [11] [14]. The problem domain size and optimal solution are denoted by $Lb \leq x_i \leq Ub$ and $f(x^*) = f(x_1 \ldots x_n)$, respectively. The symbols *Lb* and *Ub* represent lower, upper bound of the variables, respectively. As we are dealing with decimal numbers, so according to our method, x_i lies between 0 and 9, i.e. *Lb*=0 and *Ub*=9. For future reference, this lower bound and upper is denoted by Lb_d and Rb_d. So Lb_d =0 and Ub_d. =9.

If $Lb_d \geq Lb$ of the given function and $Ub_d \leq Ub$ of the given function, then value of x_i falls within the given lower and upper bound. For example, in Ackley 2 Function, *Lb* is -32 and *Ub* is 32. As Lb_d =0 and Ub_d. =9, so the value of x_i also resides within the given range

But there are so many objective functions [18] [19] where either $Lb_d < Lb$ of the given function or $Ub_d > Ub$ of the given function. For example, in Beale Function, the lower and upper bound of x_i is -4.5 and 4.5 respectively. So many other objective functions have the same criteria. In those circumstances, the lower and upper bound is mapped accordingly using the following formula [12]:

$$x_i = Lb + \frac{x_k}{10^n - 1}\ (Ub - Lb)$$

Where *n* is the string length, x_k is the value according to our range. We can attain the following properties by using the above mapping formula:

(i) The parameters can take both positive and negative values.
(ii) By using strings of different lengths, the parameters may have different exactitudes.

The initial value to optimize an equation is taken as 0 and number of 0's depends on the number of variables of the function. For example, we consider the Ackley Function 2.

$$-200e^{-0.02\sqrt{x_1^2+x_2^2}}$$

The function contains two variables; x_1 and x_2. So size of the search space is 10^2=100. Hence the initial value of x_1 and x_2 is considered as 0, 0. The other values of x_1 and x_2 for optimization of the function are obtained from (0, 0) using the RLC operator. As there is 25 normal generators of string length 2, so

searching of optimum solution can be accomplished in parallel starting from those normal generators. As previously stated, if string length is 2^n, ($n>=0$), there is no exceptional generator. Hence, if the function contains 3, 5, 6, 7 or 9 variables, then searching for the optimum solution also takes place from the exceptional generators.

The following table states the values of each Benchmark functions [11] [14] [17] with their search domain, global optimum values and optimum values obtained by the proposed algorithm. Here in this table the first column depicts the name of the Benchmark functions [15] [16]. The next column contains the function itself; the third one specifies the range. The next column with the data format *f(x, y) =z* represents that the function *f* has the global optimum at the points*(x, y)* and its optimum value at that point is *z*. The last column is stating the optimum value obtained by the proposed algorithm and that point in which it is located.

Table 6:
Comparative studies of optimum value obtained by GA and Proposed Algorithm

Function Name	Function	Domain	Optimum value by Genetic Algorithm	Optimum values obtained by the proposed algorithm
Ackley Function 2	$-200e^{-0.02\sqrt{x_1^2+x_2^2}}$	$-32 \le x_i \le 32.$	$x^* = (0, 0)$, $f(x^*) = -200$	$x^* = (0, 0)$, $f(x^*) = -200$
Adjiman Function	$f(x) = \cos(x_1)\sin(x_2) - \frac{x_1}{x_2^2+1}$	$-1 \le x_1 \le 2$, $-1 \le x_2 \le 1.$	$x^* = (2, 0.10578)$, $f(x^*) = -2.02181$	$x^* = (2, 0.11111)$, $f(x^*) = -2.02175$
Bartels Conn Function	$f(x) = \lvert x_1^2 + x_2^2 + x_1x_2\rvert + \lvert\sin x_1\rvert + \lvert\cos x_2\rvert$	$-500 \le x_i \le 500$	$x^* = (0, 0)$, $f(x^*) = 1$	$x^* = (0, 0)$, $f(x^*) = 1$
Beale Function	$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$	$-4.5 \le x_i \le 4.5$	$x^* = (3, 0.5)$, $f(x^*) = 0$	$x^* = (3.5, 0.5)$, $f(x^*) = 0.39453125$
Bohachevsky 1 Function	$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$	$-100 \le x_i \le 100$	$x^* = (0, 0)$, $f(x^*) = 0$	$x^* = (0, 0)$, $f(x^*) = 0$
Bohachevsky 2 Function	$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) * 0.4\cos(4\pi x_2) + 0.3$	$-100 \le x_i \le 100$	$x^* = (0, 0)$, $f(x^*) = 0$	$x^* = (0, 0)$, $f(x^*) = 0.18$
Bohachevsky 3 Function	$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1 + 4\pi x_2) + 0.3$	$-100 \le x_i \le 100$	$x^* = (0, 0)$, $f(x^*) = 0$	$x^* = (0, 0)$, $f(x^*) = 0$
Booth Function	$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$	$-10 \le x_i \le 10$	$x^* = (1, 3)$, $f(x^*) = 0$	$x^* = (1, 3)$, $f(x^*) = 0$

Booth Function	$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$	$-10 \le x_i \le 10$	$x^* = f(1, 3)$, $f(x^*) = 0$	$x^* = f(1, 3)$, $f(x^*) = 0$
Bukin 2 Function	$f(x) = 100(x_2 - 0.01x_1^2 + 1) + 0.01(x_1 + 10)^2$	$-15 \le x_1 \le -5$ $-3 \le x_2 \le -3$	$x^* = (-10, 0)$, $f(x^*) = 0$	$x^* = (-15, -3)$, $f(x^*) = -424.75$
Camel Function – Three Hump	$f(x) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$	Subject to $-5 \le x_i \le 5$	$x^* = f(0, 0)$, $f(x^*) = 0$	$x^* = f(1.66667, -0.55554)$, $f(x^*) = 0.408664$
Chung Reynolds Function	$f(x) = \left(\sum_{i=1}^{D} x_i^2\right)^2$	$-100 \le x_i \le 100$	$x^* = (0,\ldots 0)$, $f(x^*) = 0$	$x^* = (0,\ldots 0)$, $f(x^*) = 0$
Colville Function	$f(x) = 100(x_1 - x_2^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1)$	Subject to $-10 \le x_i \le 10$	$x^* = f(1, \cdots, 1)$, $f(x^*) = 0$	$x^* = f(1, \cdots, 1)$, $f(x^*) = 0$
Cube Function	$f(x) = 100(x_2 - x_1^3)^2 + (1 - x_1)^2$	Subject to $-10 \le x_i \le 10$.	$x^* = f(-1, 1)$, $f(x^*) = 0$	$x^* = f(1, 1)$, $f(x^*) = 0.0030$
Deckkers-Aarts Function	$f(x) = 10^5x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4$	$-20 \le x_i \le 20$	$x^* = f(0,\pm 15)$ $f(x^*) = -24777$	$x^* = f(0,9)$ $f(x^*) = -6049.53279$
FON1	$f_1(x) = 1 - exp\left(-\sum_{i=1}^{3}\left(x_i - \frac{1}{\sqrt{3}}\right)^2\right)$	$-4 \le x_i \le 4$	$x_1=x_2=x_3$	$x^* = f(0.444, 0.444, 0.444)$ $f(x^*) = 0.05161228$
FON2	$f_1(x) = 1 - exp\left(-\sum_{i=1}^{3}\left(x_i + \frac{1}{\sqrt{3}}\right)^2\right)$	$-1/\sqrt{3} \le x_i \le 1/\sqrt{3}$	$x_1=x_2=x_3$	$x^* = f(-0.444, -0.444, -0.444)$ $f(x^*) = 0.05161228$

Himmelblau Function	$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$	Subject to $-5 \le x_i \le 5$	$x^* = f(3, 2)$, $f(x^*) = 0$	$x^* = f(-2.77, 2.77)$, $f(x^*) = 4.5069$
Powell Singular Function	$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$	$-10 \le x_i \le 10$	$x^* = f(0, \cdots, 0)$, $f(x^*) = 0$	$x^* = f(0, \cdots, 0)$, $f(x^*) = 0$
Powell Sum Function	$f(x) = \sum_{i=1}^{D} \lvert x_i \rvert^{i+1}$	$-1 \le x_i \le 1$	$x^* = f(0, \cdots, 0)$, $f(x^*) = 0$	$x^* = f(0, \cdots, 0)$, $f(x^*) = 0$
Price3	$f(x) = 100(x_2 - x_1^2)^2 + 6[6.4(x_2 - 0.5)^2 - x_1 - 0.6]^2$	$-500 \le x_i \le 500$	$x* = f(\{-5,-5\},\{-5, 5\}, \{5,-5\}, \{5, 5\})$, $f(x*) = 0$	$x^* = f(1, 1)$, $f(x^*) =$ 7.395570986446986E-32
Price4	$f(x) = (2x_1^3 x_2 - x_2^3)^2 + (6x_1 - x_2^2 + x_2)^2$	$-500 \le x_i \le 500$	$x^* = f(\{0, 0\},\{2, 4\}, \{1.464,-2.506\})$, $f(x^*) = 0$	$x^* = f(0, 0)$, $f(x^*) = 0$ $x^* = f(2, 4)$, $f(x^*) = 0$
Quadratic Function	$f(x) = -3803.84 - 138.08x_1 - 232.92x_2 + 128.08x_1^2 + 203.64x_2^2 + 182.25x_1x_2$	$-10 \le x_i \le 10$	$x^* = f(0.19388, 0.48513)$, $f(x^*) = -3873.7243$	$x^* = f(0, 1)$, $f(x^*) = -3833.1200$
Rotated Ellipse 2 Function	$f(x) = x_1^2 - x_1x_2 + x_2^2$	$-500 \le x_i \le 500$	$x^* = f(0, 0)$, $f(x^*) = 0$	$x^* = f(0, 0)$, $f(x^*) = 0$
Rotated Ellipse Function	$f(x) = 7x_1^2 - 6\sqrt{3}x_1x_2 + 13x_2^2$	$-500 \le x_i \le 500$	$x^* = f(0, 0)$, $f(x^*) = 0$	$x^* = f(0, 0)$, $f(x^*) = 0$
Rump Function	$f(x) = (333.75 - x_1^2)x_2^6 + x_1^2(11x_1^2x_2^2 - 121x_2^4 - 2) + 5.5x_2^8 + \frac{x_1}{2 + x_2}$	$-500 \le x_i \le 500$	$x^* = f(0, 0)$, $f(x^*) = 0$	$x^* = f(9, 0)$, $f(x^*) =$-157.5
Styblinski-Tang Function	$f(x) = \frac{1}{2}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i)$	Subject to $-5 \le x_i \le 5$	$x^* = f(-2.903534,-2.903534)$, $f(x^*) = -78.332$	$x^* = f(-2.7777,-2.7777,-2.7777,-2.7777)$, $f(x^*) =$ -155.61652

Trecanni Function	$f(X)=x^4{}_1+4x^3{}_1+4x^2{}_1+x^2{}_2$	$-5 \le x_i \le 5$.	$x^* = f(\{0, 0\},\{-2, 0\})$, $f(x^*) = 0$	$x^* = f(\{-2.77777777777777,-0.55555555555554\}, \{-2.77777777777777, 0.55555555555554\})$, $f(x^*) = -36.9989$
Wayburn Seader 1 Function	$f(x) = (x_1^6 + x_2^4 - 17)^2 + (2x_1 + x_2 - 4)^2$	$-500 \le x_i \le 500$	$x^* = f(\{1, 2\},\{1.597, 0.806\})$, $f(x^*) = 0$	$x^* = f(1, 2)$, $f(x^*) = 0$
Wayburn Seader 2 Function	$[1.613 - 4(x_1 - 0.3125)^2 - 4(x_2 - 1.625)^2]^2 + (x_2 - 1)^2$	$-500 \le x_i \le 500$	$x^* = f\{(0.2, 1), (0.425, 1)\}$, $f(x^*) = 0$	$x^* = f(0, 1)$, $f(x^*) = 0.115685$
Wayburn Seader 3 Function	$f(x) = 2\frac{x_1^3}{3} - 8x_1^2 + 33x_1 - x_1x_2 + 5 + [(x_1 - 4)^2 + (x_2 - 5)^2 - 4]^2$	$-500 \le x_i \le 500$	$x^* = f(5.611, 6.187)$, $f(x^*) = 21.35$	$x^* = f(5, 7)$, $f(x^*) = 19.33$
Wolfe Function	$f(x) = \frac{4}{3}(x_1^2 + x_2^2 - x_1x_2)^{0.75} + x_3$	Subject to $0 \le x_i \le 2$	$x^* = f(0, \cdots, 0)$, $f(x^*) = 0$	$x^* = f(0, \cdots, 0)$, $f(x^*) = 0$
Zettl Function	$f(x) = (x_1^2 + x_2^2 - 2x_1)^2 + 0.25x_1$	$-1 \le x_i \le 5$	$x^* = f(-0.029895977602857,0)$, $f(x^*) = -0.0037912372$	$x^* = f(1,-1)$, $f(x^*) = 0.25$
Zirilli or Aluffi-Pentini's Function	$f(x) = 0.25x_1^4 - 0.5x_1^2 + 0.1x_1 + 0.5x_2^2$	Subject to $-10 \le x_i \le 10$	$x^* = (-1.0465, 0)$, $f(x^*) \approx -0.3523$	$x^* = f(1,0)$, $f(x^*) = -0.15$

Conclusion:

In this paper, our endeavor is to provide an alternative GA like parallel algorithm where only two tunable parameters are used. They are the string length (n) and number of generators. In comparison to many other meta-heuristics, much less number of tunable parameters has been used by introducing the Rotate Left and Complement (RLC) operator. It has been used to generate distinct points and also to avoid repetition of search points. We have investigated the exhaustive parallel algorithm for decimal coded string only. We have observed that $2\times n$ distinct strings are generated for an n length string. However it is also possible to implement the algorithm for any other real coded version. Here some benchmark functions have been

optimized using the discrete search points generated by the proposed algorithm. From the given results, it is observed that the proposed algorithm can produce comparable and competitive results.

References:

[1] A Samuel O. Obadan, Zenghui Wang ,"Hybrid optimization approach for complex nonlinear objective functions," *International Journal of Computing, 17(2) 2018, 102-112*

[2] Akpan, N. P. & Iwok, I.A. ,"Application of Linear Programming for Optimal Use of Raw Materials in Bakery," *International Journal of Mathematics and Statistics Invention (IJMSI) E-ISSN*: 2321 – 4767 P-ISSN: 2321 - 4759 www.ijmsi.org Volume 4 Issue 8 || October. 2016 || PP-51-57

[3] A. K.Ojha and K.K.Biswal, "Polynomial Geometric Programming Problems with Multiple Parameters" *JOURNAL OF COMPUTING, VOLUME 2, ISSUE 1*, JANUARY 2010, ISSN 2151-9617

[4] Frank, Marguerite, and Philip Wolfe, "*An Algorithm for Quadratic Programming.*" Naval Research Logistics Quarterly 3 (1956): 95-110. Web. 4 June 2015.

[5] Floudas, Christodoulos A., and V. Visweswaran., "*Quadratic Optimization*." Nonconvex Optimization and Its Applications, 2 (1995): 217-69. Web. 23 May 2015.

[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[7] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. London: John Wiley, 2001.

[8] D. Bhandari, C. A. Murthy, and S. K. Pal, "Genetic algorithm with elitist model and its convergence," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 10, no. 6, pp. 731–747, 1996

[9] B. Brey, *Intel microprocessors: 8086/8088, 80186, 80286, 80386, and 80486 architecture, programming and interfacing. New Delhi*: Prentice Hall, 1995.

[10] Momin Jamil and Xin-She Yang, *A literature survey of benchmark functions for global optimization problems*, Int. Journal of Mathematical Modelling and Numerical Optimization, Vol. 4, No. 2, pp. 150–194 (2013). DOI: 10.1504/IJMMNO.2013.055204

[11] M.J.D. Powell, An iterative method for finding stationary values of a function of several variables, Cornput. J. 5 (196'2) 147-151.

[12] Kalyanmoy Deb, *An introduction to genetic algorithms* Sadhana (1999) 24: 293. https://doi.org/10.1007/BF02823145

[13] Vedatrayee Chatterjee, Kamal Dhanda *Optimization for Best Feature Selection on Microarray gene Expression Data* , Journal of Electrical Systems, Vol. 20 No. 9s(2024) DOI**:** https://doi.org/10.52783/jes.4937

[14] Valdez, F., Castillo, O. & Peraza, C. *Fuzzy Logic in Dynamic Parameter Adaptation of Harmony Search Optimization for Benchmark Functions and Fuzzy Controllers*. *Int. J. Fuzzy Syst.* **22,** 1198–1211 (2020). https://doi.org/10.1007/s40815-020-00860-7

[15] D. I. Mattos, J. Bosch and H. H. Olsson, "*Statistical Models for the Analysis of Optimization Algorithms With Benchmark Functions*," in *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 6, pp. 1163-1177, Dec. 2021, doi: 10.1109/TEVC.2021.3081167.

[16] H. Zille and S. Mostaghim, "*Comparison study of large-scale optimization techniques on the LSMOP benchmark functions*," *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017, pp. 1-8, doi: 10.1109/SSCI.2017.8280974.

[17] S. Rajput, M. Parashar, H. M. Dubey and M. Pandit, "*Optimization of benchmark functions and practical problems using Crow Search Algorithm*," *2016 Fifth International Conference on Eco-friendly Computing and Communication Systems (ICECCS)*, 2016, pp. 73-78, doi: 10.1109/Eco-friendly.2016.7893245.

[18] Sivanandam, S., Deepa, S. (2008). Genetic Algorithm Optimization Problems. In: Introduction to Genetic Algorithms. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-73190-0_7

[19] J.G. Digalakis & K.G. Margaritis (2001) On benchmarking functions for genetic algorithms, International Journal of Computer Mathematics, 77:4, 481-506, DOI: 10.1080/00207160108805080