Walchand College of Engineering, Sangli.
(An Autonomous Institute)

Department Of
Information Technology

A Project Report
On

Design and Implementation of Rao's Algorithm on Spark Framework

submitted by

Ms. Shraddha. S. Varma (2017BTEIT00203)
Ms. Shreya. M. Bodhane (2017BTEIT00204)
Mr. Summit. B. Lohar (2017BTEIT00207)

Under the Guidance
Of

Dr. A. J. Umbarkar
Information Technology Dept,
WCE, Sangli.

Year:2019-2020

Walchand College of Engineering, Sangli.
(An Autonomous Institute)

Department Of
Information Technology

CERTIFICATE

This is to certify that the Project Report entitled, Design and Implementation of Rao's Algorithm on Spark Framework submitted by

Ms. Shraddha. S. Varma	**(2017BTEIT00203)**
Ms. Shreya. M. Bodhane	**(2017BTEIT00204)**
Mr. Summit. B. Lohar	**(2017BTEIT00207)**

to Walchand College of Engineering, Sangli, India, is a record of bonafide Project work carried out by them under my supervision and guidance and is worthy of consideration for the award of the degree of Bachelor of Technology in Information Technology of the Institute.

Dr. A. J. Umbarkar
Guide
Information Technology Dept,
WCE, Sangli.

Dr. S. P. Sonavane
Head of Department, Sangli
Information Technology Dept,
WCE, Sangli.

Acknowledgement

We feel immense pleasure in submitting this Project report entitled" Design and Implement- ation of Rao's Algorithm on Spark Framework".

We are thankful to our guide Dr. A. J. Umbarkar for their valuable guidance and kind help during completion of Project and feel great to express our sincere gratitude to other all staff members of IT Department.

We are also thankful to the Head of the 'Department of Information Technology' Dr. S. P. Sonavane for their valuable guidance during the completion of Project.

We would like to thank all faculty members and staff of Department of Information Technology for their generous help in various ways for the completion of this thesis.

We would like to thank all our friends and especially our classmates for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious. we have enjoyed their companionship so much during our stay at WCE, Sangli.

Declaration

We hereby declare that work presented in this project report titled **" Design and Implementation of Rao" Algorithm on Spark Framework** submitted by us in the partial fulfillment of the requirement of the award of the degree of **Bachelor of Technology (B.Tech)** Submitted in the Department of **Information Technology, Walchand College of Engineering**, Sangli, is an authentic record of my project work carried out under the guidance **of (Dr. A. J. Umbarkar).**

Ms. Shraddha. S. Varma (2017BTEIT00203)
Ms. Shreya. M. Bodhane (2017BTEIT00204)
Mr. Summit. B. Lohar (2017BTEIT00207)

Date:

Place: (Signature)

Table of Contents

1.Introduction

1.1 Rao Algorithm:

As the last few years we discover a population based on absolutely meta-heuristic algorithms overwhelmed with various 'new' algorithms dependent on analogy of a couple of common marvels or conduct of creatures, angles, creepy crawlies, social orders, societies, planets, instruments, and so on.

Numerous new improvement algorithms are coming & the creators proclaim that the Rao Algorithms are 'higher' than the elective algorithms . A portion of those recently proposed calculations are biting the dust clearly as there are no takers and some have gotten accomplishment to a positive degree. Nonetheless, this sort of exploration might be considered as a danger and may not make commitments to propel the circle of streamlining. It would be better if the specialists mindfulness on growing simple advancement procedures which could give incredible responses to the unpredictable difficulties as opposed to looking out creating similitude based calculations. Keeping this point in sight, these 3 simple allegory less and algorithms one of a kind boundary less improvement calculations are created.

Rao's Algorithm is the Type of Evolutionary Algorithm which is people based iterative calculation .Such calculations require regular controlling boundaries like masses size, the assortment of structure factors and end circumstance and their own arrangement of rules exact controlling boundaries.

Be that as it may, this sort of examination might be considered as a danger and won't make a commitment to support the field of improvement . It could be better if the analysts consideration on growing simple streamlining strategies which can give successful responses to the confused issues as an option of looking out creating analogy based calculations. Keeping this point in sight, three basic allegory significantly less .Furthermore, calculation explicit boundary considerably less improvement calculations are created. Along we are executing

knapsack issues with rao set of rules. We've been hurrying to make a running usage of a two fold inference of the Rao calculation which attempts to cure advancement issues through ages of populaces that other after some time to move toward the cutting edge good and move a long way from the current most exceedingly terrible in the past people. To test this usage, it was actualized to a well known arrangement of MDMKPs (Multiple Demand Multidimensional Knapsack Problems) from the net which at that point must be changed to MDMMKPs (Multiple Demand Multidimensional Multiple decision Knapsack Problems.

1.2 Rao Algorithm with Example

1. Demonstration Rao-1 Algorithm

Rao-1 algorithm with unconstrained Benchmark function i.e. sphere Function that shows that is to find out the values of Xi which minimize value using Sphere function.

Sphere Function(Benchmark function):

$$\text{Minimize } f(x) = \Sigma_{i=1}^{n} x_i^2$$

Range of X :$-1000 \leq Xi \leq 1000$

The Way to deal with this benchmark trademark is zero for all xi estimations of 0. Presently to represent the proposed calculations, let us assume a size of 5 (for example Applicant arrangements), format factors x1 and x2 and two cycles as the end basis. The underlying populace is arbitrarily created inside the scopes of the factors and the relating estimations of the target trademark are appeared in table 1. As it's far from a minimize , the most reduced cost of f(x) is thought about as the most appropriate answer and the most noteworthy estimation of f(x) is contemplated as most noticeably terrible answer.

Table 1 Initial population

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-6	18	360	-
ii	15	44	2161	worst
iii	30	-7	949	-
iv	-8	-12	208	best
v	-12	-21	585	-

from Table 1 it could be obvious that the best answer is comparing the fourth applicant and the most noticeably terrible answer is like the subsequent competitor. Utilizing the underlying answers of Table 1 and accepting arbitrary wide assortment r1 = 0.1 for x1 and r2 = 0.5 for x2 , the estimated values of the variable for x1 and x2 are determined using Eq.(1) and put into Table 2. For instance, for the 1 st competitor, the new estimations of x1 and x 2 all through the primary emphasis are determined as demonstrated as follows.

"$X'1,1,1 = X_{1,1,1} + r_{1,1,1} (X_{1,4,1} - X_{1,2,1}) = -6 + 0.1 (-8-15) = -8.3$"

"$X'2,1,1 = X_{2,1,1} + r_{2,1,1} (X\ 2,4,1 - X\ 2,2,1) = 18 + 0.5 (12-44) = -10$".

So also, the fresh out of the plastic new estimations of and X1 & X2 for the contrary applicants are determined. Table 2 shows the spic and span estimations of x1 and x2 and the relating estimations of the goal highlight.

Table 2 "New values of the x1 & x2 and the fitness function while 1st iteration (Rao_1)"

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-8.3	-10	168.89	-
ii	12.7	16	417	-
iii	27.7	-35	1992	-
iv	-10.3	-40	1706	-
v	-14.3	-49	2605	-

Presently, the estimations of f(x) of table 1 and Table 2 are as thought about & the top notch estimations of function are mulled over and situated in the next table. This is after the 1st iteration.

Table 3 Identify Best & worst case fitness values(Rao-1)

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-8.3	-10	168.89	Best
ii	12.7	16	417	-
iii	27.7	-35	1992	-
iv	-10.3	-40	1706	-
v	-14.3	-49	2605	Worst

it can be noticeable that the best case value is comparing the 1st applicant and the most exceedingly awful arrangement is relating to the 5th competitor. In the essential emphasis, the estimation of the target highlight is advanced from 208 to 168.89 and the most noticeably terrible expense of the target include is ventured forward from 2161 to 2605. Presently, accepting arbitrary R r1 = 0.1 to r X1 and r2 = 0.5 to x2 , the fresh out box new estimates of the factors forx1 and x2 determine the utilization of Eq.(1) & are situated into Table 4.

Table 4 proposes the relating estimations of target include moreover.

Table 4 "updated values and the fitness function after second iteration (Rao1)"

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-7.7	9.5	149	-
ii	13.3	35.5	1437	-
iii	28.3	-15.5	1041	-
iv	-9.7	-20.5	514	-
v	-13.4	-29.5	1049	-

The values of f(x) of above two tables are differentiate and best case values of f(x) and put into the next . This is the end of the second iteration of the Rao-1 algorithm.

Table 5 "New values of the x1 & x2 and the based on fitness comparison at the after 2nd iteration (Rao-1)"

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-7.7	9.5	149	Best
ii	13.3	35.5	1437	Worst
iii	28.3	-15.5	1041	-
iv	-9.7	-20.5	514	-

v	-13.4	-29.5	1049	-

It tends to be found that toward the finish of second emphasis, the estimation of the target work is progressed from 168.89 to 149 and the most noticeably terrible cost of the target work is advanced from 2605 to 1437. In the event that we develop the amount of emphasess, at that point the known estimation of the goal work (for example 0) might be gotten inside next not many cycles. Additionally, it's miles to be referenced that on account of amplification work issues, the top notch value way the most extreme estimation of the target work and the counts are to be continued in like manner. Therefore, the proposed approach can address every minimization and amplification issues. This exhibit is for an unconstrained improvement issue. Be that as it may, the comparable advances can be followed on account of compelled improvement issues. The fundamental distinction is that a punishment include is utilized for infringement of every limitation and the punishment esteem is worked upon the goal work.

2. Rao-2 algorithm working

Utilizing the initial solution using Table 1, and using Random Values

R1 = 0.1 & R2 = 0.5 for variable x1

R1 = 0.6 & R2 = 0.2 for variable x2

are determined utilizing Eq.(2) and put in Table 6.

For instance, for the first up-and-comer, the new estimations of x1 and x2 during the rst cycle are determined as demonstrated as follows. Here the first competitor has collaborated with the second up-and-comer. The wellness estimation of the first up-and-comer is better than the wellness estimation of the second up-and-comer and consequently thusly the measurements exchange is from first contender to second up-and-comer.

"$X'_{1,1,1} = X_{1,1,1} + R_{1,1,1}(X_{1,4,1} - X_{1,2,1}) + R_{2,1}(X_{1,1,1} - X_{1,2,1}) = -6 + 0.1\ (-8-15) + 0.6\ (6-8) = -9.5$"

"$X'_{2,1,1} = X_{2,1,1} + R_{1,2}(X_{2,4,1} - X_{2,2,1}) + R_{2,2}(X_{2,1,1} - X_{2,2,1}) = 18 + 0.5\ (-12-44) + 0.2\ (18-12) = -8.8$"

likely, the next values of variable x1 and variable x2 for the candidates are evaluated. some random interactions considered as 1 verses 4,2 verses 5, 3 verses 2, 4 verses 1 and 5 verses 3.

Table 6 shows updated values of variable x1 and variable x2 and the respective values with fitness function.

Table 6 "values of x1 and x2 and the corresponding values with fitness function during 1 iteration (Rao-2)"

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-9.5	-8.8	167.69	-
ii	14.5	20.6	634.61	-
iii	36.7	-42.4	3127.73	-
iv	-9.1	41.2	1780.25	-
v	-25.1	-46.2	2764.45	-

Presently, the estimations of f(x) of a given 2 table are as thought about and the top notch estimations of function values are mulled over and situated in Table 7. This is after the 1st iteration.

Table 7 Identify Best & worst case fitness values(Rao-2)

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-9.5	-8.8	167.69	Best
ii	14.5	20.6	634.61	-
iii	36.7	-42.4	3127.73	-
iv	-9.1	41.2	1780.25	-
v	-25.1	-46.2	2764.45	Worst

From the above table it tends to be that the best candidate is comparing the fourth competitor and the most exceedingly awful arrangement is relating to the third up-and-comer. Presently, during the subsequent cycle, expecting irregular numbers R1 =0.1 and R2 = 0.5 for x1 and R1 = 0.6 and R2 = 0.2 for x2 , the new estimations of the factors for x1 and x2 are determined utilizing Equation (2). Here the arbitrary connections are taken as 1 versus 4, 2 versus 5, 3 versus 2, 4 versus 1 and 5 versus 3.

Table 8 "values of x1 and x2 and the respective values with fitness function during second iteration (Rao-2)"

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-7.7	3.42	70.9864	-
ii	9.7	34.18	1262.3624	-
iii	51.58	-19.14	3026.856	-

iv	-7.78	66.38	4461.524	-
v	30.5	-26.74	1645.2276	-

Evaluating Best & Worst case values

Table 9

Updated x1 and x2 and the corresponding values with fitness function at the end of second iteration (Rao-2)

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-7.7	3.42	70.9864	Best
ii	9.7	34.18	1262.3624	-
iii	51.58	-19.14	3026.856	-
iv	-7.78	66.38	4461.524	Worst
v	30.5	-26.74	1645.2276	-

From Table 9 it could be obvious that the quality arrangement is related to the second applicant and the most exceedingly awful answer is compared to the 5nd up-and-comer. It may very well be found that the cost of the target included progressing from 167.69 to 70.9864 in cycles. Additionally, the most noticeably terrible expense of the target trademark is advanced from 2764.45 to 4461.524. in only two cycles. In the event that we blast the quantity of emphasess, at that point the known cost of the goal trademark (for example 0) can be obtained inside after not many cycles. Likewise, like Rao-1, the proposed Rao-2 can adapt to each unconstrained and controlled minimization just as boost issues.

3. Demo of Rao-3 algorithm

Utilizing the initial solutions of Table 1, and using Random Values

R1 = 0.1 & R2 = 0.5 for variable x1

R1 = 0.6 & R2 = 0.2 for variable x2

are determined utilizing Eq.(3) and put in Table 10.

Here the 1st candidate has collaborated with the 4th candidate. The fitness cost of the 1st candidate is greater than the fitness cost of the 2nd candidate and subsequently the measurements change from 1st candidate to 2nd candidate.

" $X'_{1,1,1} = X_{1,1,1} + r_{1,1}(X_{1,4,1} - X_{1,2,1}) + r_{2,1}(X_{1,1,1} - X_{1,4,1}) = -6 + 0.1\ (-8-15) + 0.5\ (6+8) = -0.1$"

"$X'_{2,1,1} = X_{2,1,1} + r_{1,2}(X_{2,4,1} - X_{2,2,1}) + r_{2,2}(X_{2,1,1} - X_{2,4,1}) = 18 + 0.60\ (-12-44) + 0.20\ (18+12) = -4$"

Additionally, the new estimations of x1 and x2 for different candidates are determined and calculated. Nowonwords the arbitrary interactions are taken as 1 versus 4, 2 versus 5, 3 versus 2, 4 versus 1 and 5 versus 3. Table 10 shows the new values of x1 and x2 and the relating estimations of the objective function.

Table 10

"New estimation of the variables and the objective function during 1st iteration (Rao-3)"

Candidate	X1	X2	sphere function i.e f(X)	Status
i	0.1	-4	16.1	-
ii	28.9	29	1676	-

iii	36.7	-42.4	314.65	-
iv	-1.9	-41.2	1701.05	-
v	-25.1	43.4	2513.57	-

This finishes the 1st iteration of the Rao3 algorithm. From Table 11 it could be noticeable that the Best answer is corresponding to the 4th candidate and the worst solution is compared to the three 3rd candidates.

Table 11

"Updated x1 and x2 and the corresponding values with fitnessfunction at the end of first iteration (Rao-3)"

Candidate	X1	X2	sphere function i.e f(X)	Status
i	0.1	-4	16.1	Best
ii	28.9	29	1676	-
iii	36.7	-42.4	3144.65	worst
iv	-1.9	-41.2	1701.05	-
v	-25.1	43.4	2513.57	-

for x2 , the new values of variables for the x1 and x2 are evaluating using Equation (3). The random interaction are taken as1 versus 4, 2 versus 5, 3 versus 2, 4 versus 1 and 5 versus 3. Table 12 shows the new values of x 1 and x 2 and the relating values of the objective feature at some point of the second generation.

Table 12 "Updated x1 and x2 and the corresponding values with fitness function during second iteration (Rao-3)"

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-2.36	-19.76	396.02	-
ii	57.64	2.92	3330.89	-
iii	37.72	-62.92	5381.72	-
iv	-4.48	-55.36	3084	-
v	-35.72	37.36	2671	-

Now, the estimation of f(x) of Tables 11 and 12 are compared and the best estimation of f(x) are thought and put in Table 13. That completes the 2nd iteration of the Rao-3 algorithm.

Table 13

"Updated estimation of the variables and the objective function dependent on fitness examination toward the end of second iteration (Rao-3)"

Candidate	X1	X2	sphere function i.e f(X)	Status
i	-2.36	-19.76	396.02	Best
ii	57.64	2.92	3330.89	-
iii	37.72	-62.92	5381.72	Worst
iv	-4.48	-55.36	3084	-
v	-35.72	37.36	2671	-

From Table 13 it may be visible that the fabulous solution is corresponding to the second candidate and the worst solution is corresponding to the 5nd candidate. It can be observed that the price of the objective feature is advanced from 16.1 to 396.02 in only iterations. Similarly, the worst price of the objective characteristic is advanced from 3144.65 to 5381.72 in only iterations. If we increase in variety of iterations then the known value of the objective function (i.E.0) can be acquired with some little iteration.

Additionally, like Rao_1 , Rao_2 & the proposed Rao_3 can likewise adapt to each unconstrained and restricted minimization in addition to the maximization problems. It can be expressed that the above 3 shows with the random numbers are simply to make the reader familiar with the working of the proposed algorithms. While executing the algorithms distinct random numbers can be created for the pan of the different iterations and the computations could be accomplished in like manner.

1.3 Problem Definition

Design Implementation of Rao's Algorithm for Solving Optimization Problem i.e. in real world problems.

1.4 Objectives

1. To Design spark based models of Rao's Algorithm.
2. To Implement Rao's Algorithm in a sequential and Distributed Manner.
3. Apply Rao's Algorithm to solve Optimization Problems.

1.5 Relevance

The Rao Algorithm should be tried for unconstrained(without some condition) & constrained engineering optimization problems.

2. Optimization Problem Formulation

Let f(y) the target capacity to be limited (or augmented). At any emphasis i, accept that there are 'm' scope of format factors, 'n' amount of up-and-comer arrangements (for example masses size, k=1,,...n). Let best competitor top notch gets the acceptable expense of f(y) (for example F(x)satisfactory) in the entire competitor arrangements and the most noticeably terrible applicant most noticeably awful acquires the most noticeably awful cost of f(y) (for example F(y)worst) inside the entire applicant arrangements. In the event that Y,k,i is the cost of the jth variable for the kth applicant at some phase in the ith new discharge, at that point this cost is changed according to the resulting conditions.

"Y'j,k,i= Yj,k,i+ r1,j,i (Yj,best,i -Yj,worst,i)" …..(1)

"Y'j,k,i= Yj,k,i+ r1,j,i (Yj,best,i -Yj,worst,i) + r2,j,i (Yj,k,i or Yj,l,i- Yj,l,i or Yj,k,i)" .(2)

"Y'j,k,i= Yj,k,i+ r1,j,i (Yj,best,i -Yj,worst,i) + r2,j,i (Yj,k,i or Yj,l,i-(Yj,l,i Yj,k,i))" .(3)

where, Yj,best,i is the estimation of the variable j for the best candidate and Yj,worst,i is the estimation of the variable j for the most exceedingly awful applicant during the ith cycle. Y'j,k,i is the refreshed estimation of Yj,k,i and r1,j,i and r2,j,i are the two arbitrary numbers for the jth variable during the ith emphasis in the range [0, 1]. In Eqs.(2) and (3), the term Yj,k,i or Yj,l,i demonstrates that the candidate arrangement k is contrasted and any haphazardly picked applicant arrangement l and the data is traded dependent on their wellness esteems. On the off chance that the wellness estimation of kth arrangement is better than the wellness estimation of lth arrangement then the term Yj,k,i or Yj,l,i" becomes Yj,k,i. Then again, if the wellness estimation of lth arrangement is better than the wellness estimation of kth arrangement then the term Yj,k,i or Yj,l,i" becomes Yj,l,i. Correspondingly, if the wellness estimation of kth arrangement is better than the wellness estimation of lth arrangement then the term Yj,l,i or

Yj,k,i" becomes Yj,l,i. In the event that the wellness estimation of lth arrangement is better than the wellness estimation of kth arrangement then the term Yj,l,i or Yj,k,i" becomes Yj,k,i.

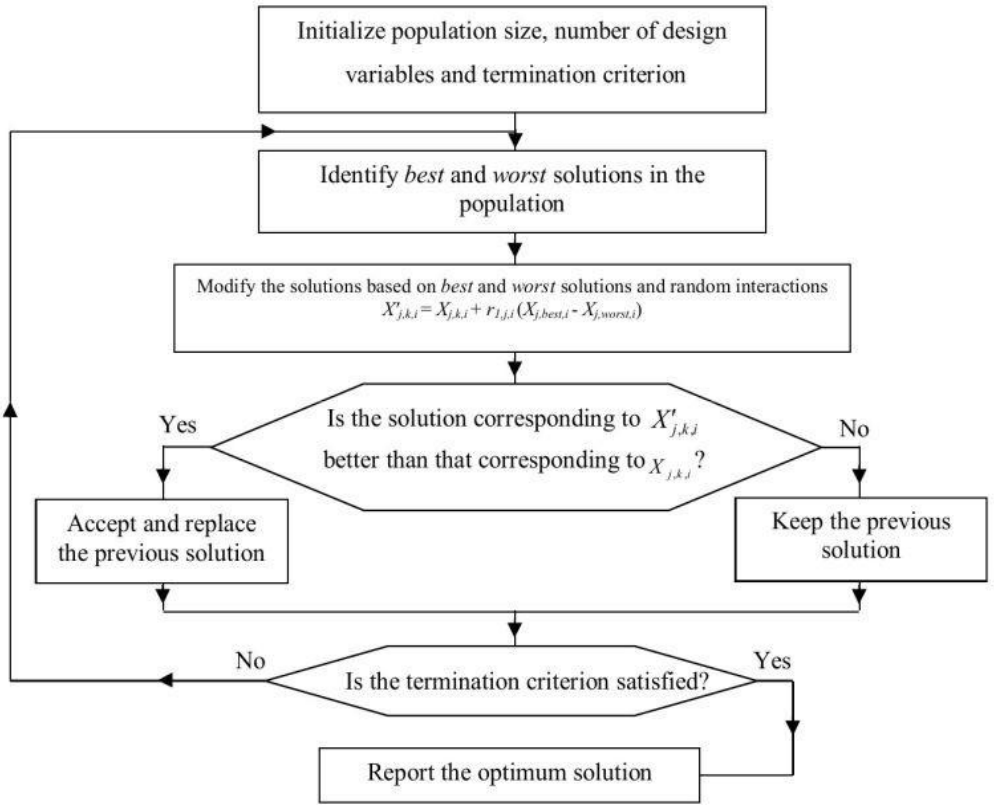

Fig1 : Rao flowchart

1. Design Variables

The formulation of an optimization problem starts with identification of design variables, which vary inside the course of the optimization process. A layout problem commonly includes a many layout parameters, of which some are especially sensitive to the right running of the layout. These parameters are referred to as layout variables. In relation to the format variable other design parameters commonly remain constant or vary.

2. Constraints

After selecting the design variables, the next work is to discover the constraints associated with the OP. The constraint represents a few functional relationships, most of the layout variables and other design parameters satisfying certain physical phenomena and sure resource limitations. Some of these issues require that the design continue to be in static or dynamic equilibrium.There is but no unique manner to formulate a constraint in all problems. The no of constraints and the character that must be included within the system is dependent on the user.

3. Objective function

The common engineering objectives contain minimization of all price of manufacturing, maximization of internet earnings earned. Although the maximum of the above objectives may be quantified easily, there are some goals that might not be quantified easily.

Moreover, in any real-global optimization problem, the designer may need to optimize simultaneously a couple of objectives. The fashion designer chooses the most vital objective as the goal feature of the optimization problem, and the opposite targets are considered as constraints where the values are restricted within a particular range. The objective can be of two sorts both the goal characteristic is to be maximized or it needs to be minimized.

4. Variable bounds

Finally, we set the maximum and minimum bounds on every design variable. Givn problems, the constraints absolutely surround the viable region.

Mentioned above 4 duties are completed, the OP can be mathematically written in a unique format, called nonlinear programming (NLP) format.

3. Literature Review

- ## Rao Algorithm

Rao Algorithm is proposed by Ravipudi Venkata Rao in a from "S.V. National Institute of Technology, Ichchanath, Surat, India." He expressed "Rao algorithms: Three metaphor-less simple algorithms for solving optimization problems" Algorithms depend on the best and worst arrangements got during the improvement procedure and the random cooperations between the competitor arrangements.

These 3 algorithms depend on the high-caliber and most exceedingly terrible arrangements in the masses and the irregular communications between the applicant arrangements. Much the same as TLBO algorithms and Jaya algorithms Rao algorithms no longer require any calculation specific boundaries and in this way the architect's weight to advise the calculation explicit boundaries to get the best results is wiped out.

Apache Spark is a lightning-fast bunch figuring innovation, intended for sure fire computation. It depends on Hadoop Map Reduce and it stretches out the Map-decrease model to productively utilize it for additional sorts of calculations, which incorporates intuitive inquiries and flow handling. The primary capacity of Spark is its in-memory group registering that will speed up a product. Flash is intended to cover a wide assortment of outstanding burdens which incorporates clump bundles, iterative calculations, intuitive inquiries and spilling. Aside from supporting a ton of these remaining task at hand in a particular framework, it lessens the administration weight of protecting separate devices.

- ## Evolution of Apache Spark

1. Spark is one in each Hadoop's sub undertaking advanced in 2009 in UC Berkeley's AMPLab by Matei Zaharia. Apache Spark has the accompanying highlights.

2. Speed Spark permits to run programming in a Hadoop group, up to one hundred times speedier in memory, and multiple times quicker when strolling on plate. This is conceivable with the guide of lessening the assortment of read/compose tasks to circle. It stores the middle of the road preparing records in memory.

3. Supports different dialects Spark presents worked in API in Java, Scala, or Python. In this manner, you can compose applications in exceptional dialects. Spark thinks of eighty serious extent administrators for intelligent questioning.

4. Advanced Analytics Spark not most effectively helps 'Map' and 'reduce'. It also supports SQL queries, Streaming facts, Machine learning (ML), and Graph algorithms.

- **Self-adaptive Multi-population Rao Algorithms**

Rao algorithms are enhanced with the multi-populace explore task to beautify the variance of search. The variety of sub-populace is modified adaptively thinking about energy of outcomes. to control the exploration and exploitation of the explore task.

The features of multi-populace optimization strategies are useful because of the subsequent reasons:

1. In the long term the diversity of the explore task may be maintained by allocating the entire populace into groups. because numerous sub-populace may be located in dissimilar areas of the trouble search space & are having the capability to explore in diverse areas simultaneously.
2. Populace based optimization methods may be without difficulty included into different methods.The number of subpopulace relies upon the intricacy of the problem. Hence, it's far very hard to determine the wide variety of sub-populace for the effective processing of the algorithm. The incorrect scale of sub populace may lose the scale of the search procedure. In sequence to keep away from those matters, the gift work

suggested self-adaptive multi-populace(SAMP)algorithms to comprise the leads of multi-population outlook in simple Rao algorithms. The SAMP-Rao algorithms flexibly exchange the quantity of sub-populace primarily based at trade in the best fitness value in every iteration that allows to manipulate the exploration & exploitation prices of search technique.

(i) To suggest SAMP-Rao algorithms that cut up the entire populace into the scale of sub-populace to upgrade the variety of search & change the number of subpopulace adaptively primarily based on the best value of fitness feature in each iteration to control the exploration & the exploitation of the quest process.

(ii) To look over the performance of the proposed algorithms.

4. SAMP-Rao Algorithms

The course of looking for most efficient solution the usage of Rao algorithms depends on the best and worst candidate outcomes inside the complete populace and the random interactions among the candidate outcomes. Let f1 is the fitness function that's to be maximized (or minimized). At any generation a, assume that there are 'm' wide variety of populace (i.e., candidate answers, z= 1,2, ..., m) and 'o' wide variety of design variables.

Let the quality cost of fitness characteristic f1 (i.e. f1 first-rate) gained with the best candidate variables,i.e first-rate in complete candidate outcomes and the worst value of f1 (i.e. f1 worst) obtained with the worst candidate, i.e., worst within the whole populace. If Y z,x,a is the cost of the xth variable for the zth candidate during the ath new release, then price is updated for the use of anyone of subsequent three equations.

The primary objectives of this work are:

$$Y'_{z,x,a} = Y_{z,x,a} + r1_{z,x,a}(Y_{best,x,a} - Y_{worst,x,a}) \quad \text{.....[1]}$$

$$Y'_{z,x,a} = Y_{z,x,a} + r11_{z,x,a}(Y_{best,x,a} - Y_{worst,x,a}) + r12_{z,x,a}(|Y_{z,x,a} \text{ or } Y_{Z,x,a}| - |Y_{Z,x,a} \text{ or } Y_{Z,x,a}|) \quad \text{.....[2]}$$

$$Y'_{z,x,a} = Y_{z,x,a} + r11_{z,x,a}(Y_{best,x,a} - |Y_{worst,x,a}|) + r12,_{z,x,a}(|Y_{z,x,a} \text{ or } Y_{Z,x,a}| - (Y_{Z,x,a} \text{ or } Y_{z,x,a}) \quad \text{..... [3]}$$

Where, Y(best,x,a) is price of high-quality candidate for the variable x & Y(worst,x,a) is the value of the worst candidate for the variable x all through the ath iteration. Y'(u,v,w) is updated price of Y(z,x,a) and r11(z,x,a) and r12(z,x,a)are the 2 random numbers for the zth candidate of xth variable for the duration of the ath iteration inside the scale [0, 1].

In Equations (2) and (3), the term Y(z,x,a) or Y(Z,x,a) indicates that the zth candidate answer

is in comparison with any randomly picked Zth candidate answer & the records are swap on the premise in their fitness function values. If the fitness feature price of an zth candidate answer is better than the fitness feature value of Zth candidate outcome then the term "Y(z,x,a) or Y(z,x,a) " turns into Y(z,x,a) and the term "Y(Z,x,a)or Y(z,x,a)" will become Y(Z,x,a) . On the alternative hand, if the fitness feature cost of an Zth candidate answer is best than the fitness characteristic value of uth candidate outcome then the time period "Y (z,x,a)or Y (Z,x,a) " turns into Y(Z,x,a) and the time period "Y(Z,x,a) or Y(z,x,a)" turns into Y(z,x,a) The searching system of worldwide top-rated is carried out by the usage of Eq Rao-1 algorithm, Rao-2algorithm, and Rao-3 algorithm.

Just like the TLBO algorithm and Jaya algorithm Rao algorithms are also unbiased of algorithm-specific parameters and therefore the efforts of designers are reduced to tune the set of rules-unique parameters for obtaining the exceptional outcomes. Also,the proposed algorithms are comparatively straightforward & simpler .

- **In SAMP-Rao algorithms, the following modifications are made to basic**

Rao algorithms:

(i) The SAMP-Rao algorithms make use of some subpopulations with the aid of cutting the total population into the quantity of agencies based on first-class outcomes. The usage of the wide variety of sub-populations roll out the answers over a search space as opposed to focusing on selected area. Therefore,the proposed algorithms are predicted to reach a foremost outcome.

(ii) SAMP-Rao algorithms reform the number of sub-populations adaptively at some stage in the search manner based totally on exceptional of health value. It means that the quantity of

sub-populace will be expanded or decline. This trait supports the search method for searching most useful outcomes and for enhancing the diversification the search manner. Furthermore, duplicate outcomes are restored with the aid of newly produced outcomes to preserve range & to beautify the exploration procedure.

The flow diagram of the SAMP-Rao1 arrangement of rules outlined in Figure :

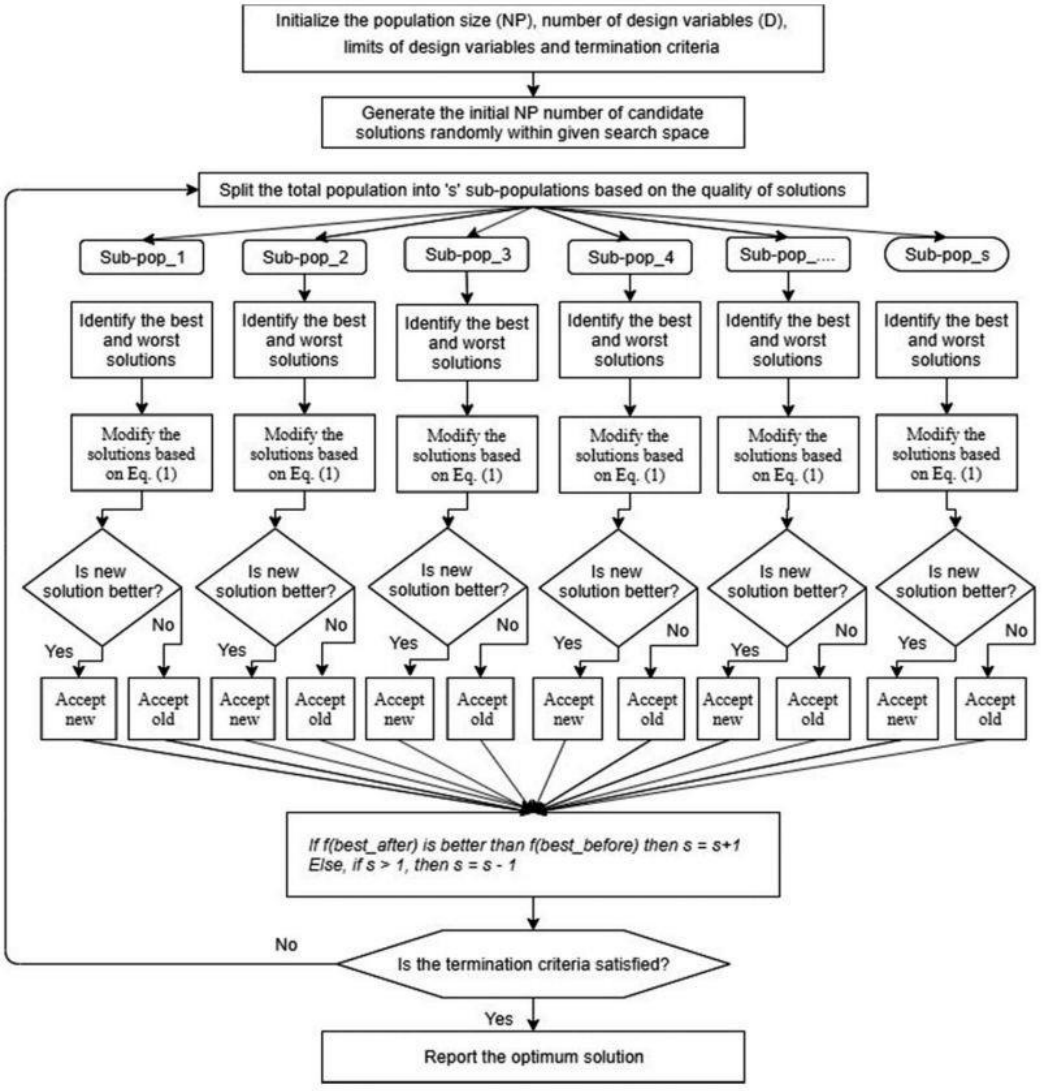

Fig 2: Flow chart of Samp

Stage 1: At first set the amount of populaces (p), the scope of plan factors (d), &
end model (for the current work, end rule is most wide assortment of highlight evaluations(maxfes))

Stage 2: Generate the irregular primer applicant arrangements inside the hunt space.

Stage 3: Split up the whole masses into 'S' number of organizations dependent on the high-caliber of the outcomes.(to start with S = 2).

Stage 4: Apply one of the Rao calculations for customizing the arrangements in every sub-populace bunch freely. Contrast each changed answer and the comparing vintage reply and get if and just in the event that it is best than the vintage answer.

Stage 5: Integrate the entire sub-populaces together. Check whether f1 (superb previously) is best than f1(excellent after). Here, f1(satisfactory previously) is past remarkable arrangement of entire people and f1(exceptional after) is the current top notch arrangement inside the whole masses. On the off chance that the expense of f1 (nice after) is best than the expense of f1(first-class previously), S is expanded by 1 (S = S + 1) to development for investigation normal look for way. Something else, S is diminished by method of 1 (S = S- 1) on the grounds that the arrangement of rules wants to be more exploitative than explorative.

Stage 6: Check the end standard. On the off chance that the look for process has arrived at the most assortment of highlight assessments, at that point end the circle and record the top notch top of the line arrangement. Something else, follow the resulting steps:
(I) For re-partitioning the people, visit Step 3. (ii) Replace the copy answers with arbitrarily produced arrangements

4. Design & Methodology:

4.1 Design of Rao Algorithm:

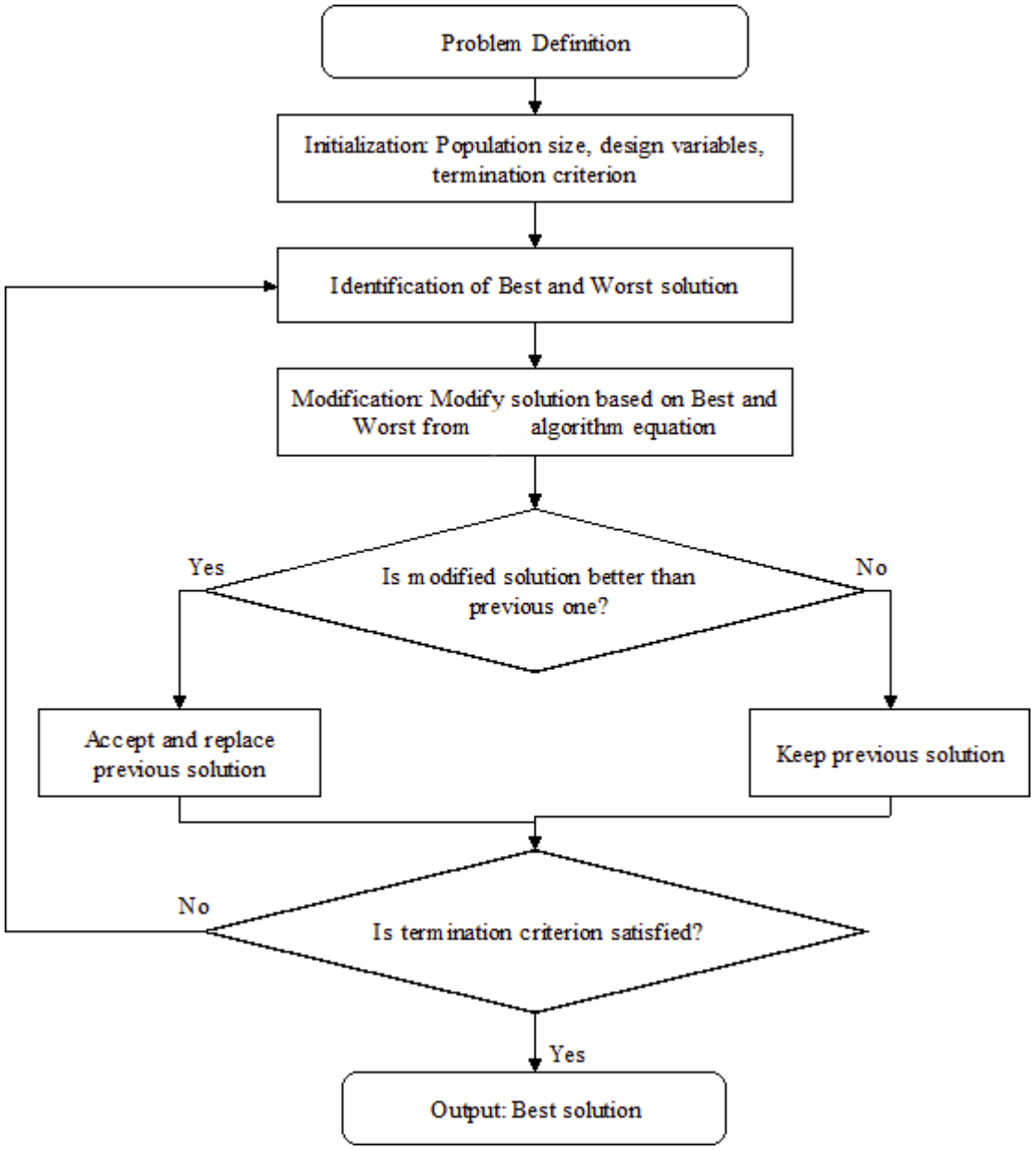

Fig 3: Flow design of Rao Algorithm

4.2 Algorithms and Pseudo Code:

- **Rao's Algorithm :**

"""-*- coding: utf-8 -*-

RAO_algorithm.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/github/sbl003/BtechProject/blob/sblbranch/RAO_algorithm.ipynb

"""

```
import random
import math
import numpy as np
maxfes = 5000 #Maximum functions evaluation
dim = 30 #Number of design variables
SearchAgents_no = 10 #Population size
Max_iter = math.floor(maxfes/SearchAgents_no) #Maximum number of iterations
lb = -100*np.ones(dim) #lower bound
ub = 100*np.ones(dim) #upper bound

"""Objective function
```

```
   Sphere function
"""

def fitness(particle):
  y = 0
  for i in range(dim):
    y= y + particle[i]**2 # sphere function
  return y

"""RAO 1 equation
   X'j,k,i = Xj,k,i + r1,j,i (Xj,best,i - Xj,worst,i)
"""

Positions = np.zeros((SearchAgents_no, dim)) # search agent position
best_pos = np.zeros(dim) # search agent's best position
worst_pos = np.zeros(dim) # search agent's worst position
finval = np.zeros(Max_iter) # best score of each iteration
f1 = np.zeros(SearchAgents_no) # function value of current population
f2 = np.zeros(SearchAgents_no) # function value of updated population
for i in range(dim):
  Positions[:, i] = np.random.uniform(0,1, SearchAgents_no) * (ub[i] - lb[i]) + lb[i]
for k in range(0,Max_iter):
  best_score = float("inf")
  worst_score = float("-inf")
  for i in range(0,SearchAgents_no):
    # Return back the search agents that go beyond the boundaries of the search space
```

```
    for j in range(dim):
      Positions[i,j]=np.clip(Positions[i,j], lb[j], ub[j])
      f1[i]= fitness(Positions[i,:])
      if f1[i] < best_score :
        best_score=f1[i].copy(); # Update best
        best_pos=Positions[i,:].copy()
      if f1[i] > worst_score :
        worst_score=f1[i].copy(); # Update worst
        worst_pos=Positions[i,:].copy()
    # Update the Position of search agents including omegas
    finval[k] = best_score
    print("The best solution is: ",best_score , " in iteration number: ",k+1)
    Positioncopy = Positions.copy()
    for i in range(0,SearchAgents_no):
      for j in range (0,dim):
        r1=random.random() #''' r1 is a random number in [0,1]'''
        Positions[i,j]= Positions[i,j] + r1*(best_pos[j]-worst_pos[j]) #change in position
        Positions[i,j]=np.clip(Positions[i,j], lb[j], ub[j])
      f2[i] = fitness(Positions[i,:])
  for i in range(0,SearchAgents_no):
    if (f1[i] < f2[i]):
      Positions[i,:] = Positioncopy[i,:]
best_score = np.amin(finval)
print("The best solution is: ",best_score)


"""Rao-2 algorithm:
```

X'j,k,i = Xj,k,i + r1,j,i (Xj,best,i - Xj,worst,i) + r2,j,i (| Xj,k,i or Xj,l,i | - | Xj,l,i or Xj,k,i |)

"""

```
Positions = np.zeros((SearchAgents_no, dim)) # search agent position
best_pos = np.zeros(dim) # search agent's best position
worst_pos = np.zeros(dim) # search agent's worst position
finval = np.zeros(Max_iter) # best score of each iteration
f1 = np.zeros(SearchAgents_no) # function value of current population
f2 = np.zeros(SearchAgents_no) # function value of updated population
for i in range(dim):
  Positions[:, i] = np.random.uniform(0,1, SearchAgents_no) * (ub[i] - lb[i]) + lb[i]
for k in range(0,Max_iter):
  best_score = float("inf")
  worst_score = float("-inf")
  for i in range(0,SearchAgents_no):
    # Return back the search agents that go beyond the boundaries of the search space
    for j in range(dim):
      Positions[i,j]=np.clip(Positions[i,j], lb[j], ub[j])
      f1[i]= fitness(Positions[i,:])
      if f1[i] < best_score :
        best_score=f1[i].copy(); # Update best
        best_pos=Positions[i,:].copy()
      if f1[i] > worst_score :
        worst_score=f1[i].copy(); # Update worst
        worst_pos=Positions[i,:].copy()
    # Update the Position of search agents including omegas
    finval[k] = best_score
```

```
    print("The best solution is: ",best_score , " in iteration number: ",k+1)
    Positioncopy = Positions.copy()
    r = np.random.randint(SearchAgents_no, size=1)
    for i in range(0,SearchAgents_no):
      if (f1[i] < f1[r]):
        for j in range (0,dim):
          r1=random.random() # ‘’’r1 is a random number in [0,1]’’’
          r2=random.random() # ‘’’r1 is a random number in [0,1]’’’
                                  Positions[i,j]=  Positions[i,j]  +  r1*(best_pos[j]-worst_pos[j])  +
r2*(np.abs(Positions[i,j])-np.abs(Positions[r,j]))#change in position
          Positions[i,j]=np.clip(Positions[i,j], lb[j], ub[j])
      else :
        for j in range (0,dim):
          r1=random.random() # r1 is a random number in [0,1]
          r2=random.random() # r1 is a random number in [0,1]
                                  Positions[i,j]=  Positions[i,j]  +  r1*(best_pos[j]-worst_pos[j])  +
r2*(np.abs(Positions[r,j])-np.abs(Positions[i,j]))#change in position
          Positions[i,j]=np.clip(Positions[i,j], lb[j], ub[j])
      f2[i] = fitness(Positions[i,:])
    for i in range(0,SearchAgents_no):
      if (f1[i] < f2[i]):
        Positions[i,:] = Positioncopy[i,:]
best_score = np.amin(finval)
print("The best solution is: ",best_score)


"""Rao-3 algorithm:
```

X'j,k,i = Xj,k,i + r1,j,i (Xj,best,i - | Xj,worst,i |) + r2,j,i (| Xj,k,i or Xj,l,i | - (Xj,l,i or Xj,k,i))

"""

```
Positions = np.zeros((SearchAgents_no, dim)) # search agent position
best_pos = np.zeros(dim) # search agent's best position
worst_pos = np.zeros(dim) # search agent's worst position
finval = np.zeros(Max_iter) # best score of each iteration
f1 = np.zeros(SearchAgents_no) # function value of current population
f2 = np.zeros(SearchAgents_no) # function value of updated population
for i in range(dim):
  Positions[:, i] = np.random.uniform(0,1, SearchAgents_no) * (ub[i] - lb[i]) + lb[i]
for k in range(0,Max_iter):
  best_score = float("inf")
  worst_score = float("-inf")
  for i in range(0,SearchAgents_no):
    # Return back the search agents that go beyond the boundaries of the search space
    for j in range(dim):
      Positions[i,j]=np.clip(Positions[i,j], lb[j], ub[j])
      f1[i]= fitness(Positions[i,:])
      if f1[i] < best_score :
        best_score=f1[i].copy(); # Update best
        best_pos=Positions[i,:].copy()
      if f1[i] > worst_score :
        worst_score=f1[i].copy(); # Update worst
        worst_pos=Positions[i,:].copy()
    # Update the Position of search agents including omegas
    finval[k] = best_score
```

```
    print("The best solution is: ",best_score , " in iteration number: ",k+1)
    Positioncopy = Positions.copy()
    r = np.random.randint(SearchAgents_no, size=1)
    for i in range(0,SearchAgents_no):
      if (f1[i] < f1[r]):
        for j in range (0,dim):
          r1=random.random() # r1 is a random number in [0,1]
          r2=random.random() # r1 is a random number in [0,1]
                          Positions[i,j]= Positions[i,j] + r1*(best_pos[j]-np.abs(worst_pos[j])) +
r2*(np.abs(Positions[i,j])-Positions[r,j])#change in position
          Positions[i,j]=np.clip(Positions[i,j], lb[j], ub[j])
      else :
        for j in range (0,dim):
          r1=random.random() # r1 is a random number in [0,1]
          r2=random.random() # r1 is a random number in [0,1]
                          Positions[i,j]= Positions[i,j] + r1*(best_pos[j]-np.abs(worst_pos[j])) +
r2*(np.abs(Positions[r,j])-Positions[i,j])#change in position
          Positions[i,j]=np.clip(Positions[i,j], lb[j], ub[j])
      f2[i] = fitness(Positions[i,:])
    for i in range(0,SearchAgents_no):
      if (f1[i] < f2[i]):
        Positions[i,:] = Positioncopy[i,:]
best_score = np.amin(finval)
print("The best solution is: ",best_score)
```

- **THE BENCHMARKS CONSIDERED**

1. Himmelblau function

```
import math
import numpy as np
import random
import pandas as pd
def myobj (p1):
F=[]
for i in range (len(p1)):
x=p1.loc[i]
f=(((x[0]**2)+x[1]-11)**2)+((x[0]+x[1]**2)-7)**2
F.append(f)
return F

pop size = 25
Gen = 1000
lb=[-5,-5]
ub=[5,5]
def initialpopulation(mini,maxi,pop size):
pop=[]
for i in range(pop size):
p=[]
for a,b in zip(mini,maxi):
p.append(a + (b-a) * random.random())
pop.append(p)
ini pop=pd.DataFrame(pop)
```

```
return ini pop
def updatepopulation(p1,dim):
best x = np.array(p1.loc[p1[ 0 f 0 ].idxmin][0 : dim])
worst x = np.array(p1.loc[p1[ 0 f 0 ].idxmax][0 : dim])
new x = []
for i in range(len(p1)):
old x = np.array(p1.loc[i][0 : dim])
r1=np.random.random(dim)
r2=np.random.random(dim)
new x .append(old x + r1 ∗ (best x − abs(old x )) − r2 ∗ (worst x − abs(old x )))
new x .append(old x + r1 ∗ (best x − worst x ))
new p 1 = pd.DataFrame(new x )
return new p 1


def greedyselector(p1,new p 1) :
for i in range(len(p1)):
if p1.loc[i]['f']¿new p 1.loc[i][ 0 f 0 ] :
p1.loc[i]=new p 1.loc[i]
return p1
def trimr(new p 1, lb, ub) :
col=new p 1.columns.values
for i in range(len(new p 1)) :
for j in range(len(col)):
if new p 1.loc[i][j] > ub[j] :
new p 1.loc[i][j] = ub[j]
elif new p 1.loc[i][j] < lb[j] :
```

```
new p 1.loc[i][j] = lb[j]
return new p 1
def Rao1(*argv):
pop size, Gen, mini, maxi= argv
lb=np.array(mini)
ub=np.array(maxi)
p1=initialpopulation(lb,ub,pop s ize)
p1['f']=myobj(p1)
dim=len(lb)
gen=0
best=[]
while (gen¡Gen):
new p 1 = updatepopulation(p1, dim)
new p 1 = trimr(new p 1, lb, ub)
new p 1[ 0 f 0 ] = myobj(new p 1)
p1=greedyselector(p1,new p 1)
gen=gen+1
print(gen)
best=p1['f'].min()
xbest=p1.loc[p1['f'].idxmin()][0:dim].tolist()
print('Best='.format(best))
print('xbest='.format(xbest))
return best,xbest
best,xbest = Rao1(pop size, Gen, lb, ub)
print('The objective function value = '.format(best))
print('The optimum values of variables = '.format(xbest))
```

2. Optimization Function

```
import numpy as np
from random import random, randint, uniform
import copy
import sys
from math import cos, exp, sqrt
def fitness( x, dimensions, f ):
if f == 1:
ans = -20 * exp( -0.2 * sqrt( 0.5*(x[0]**2 + x[1]**2) ) ) - exp( 0.5*(
cos(2*3.14*x[0]) + cos(2*3.14*x[1]) ) ) + exp(1) + 20
return ans
if f == 2: ans = ( 1.5 - x[0] + x[0]*x[1] )**2 + ( 2.25 - x[0] + x[0]*(x[1]**2) )**2
+ (2.625 - x[0] + x[0]*(x[1]**3))**2
return ans
if f == 3:
ans = ( x[0] + 2*x[1] - 7)**2 + ( 2*x[0] + x[1] - 5)**2
return ans
if f == 4:
ans = -cos(x[0])*cos(x[1])*exp( -((x[0]-3.14)**2 + (x[1]-3.14)**2))
return ans
if f==5:
ans = 0.26*( x[0]**2 + x[1]**2 ) - 0.48*x[0]*x[1]
return ans
if f == 6:
s=0
for i in range(dimensions):
s = s + x[i]**2 - 10*cos(2*3.14*x[i])
```

```
return 10*dimensions + s
if f==7: s = 0
for i in range(dimensions-1):
s = s + 100 * ( x[i+1] - x[i]**2 ) ** 2 + ( 1 - x[i] )**2
return s
else:
return 0
def Rao1 opt(f):
gens = 100
pop size = 40
if f == 1:
dimensions = 2
min value = -5
max value = 5 if f == 2: dimensions = 2
min value = -4.5
max value = 4.5
if f == 3: dimensions = 2
min value = -10
max value = 10
if f == 4: dimensions = 2
min value = -100
max value = 100
if f == 5: dimensions = 2
min value = -10
max value = 10
if f == 6:
dimensions = 2
```

```
min value = -5.12
max value = 5.12
if f == 7:
dimensions = 10
min value = -( pow( 2, 63) - 1 )
max value = pow( 2, 63 ) -1
population = np.zeros(( pop size, dimensions ))
for i in range(pop size):
for j in range(dimensions):
population[i][j] = uniform( min value, max value )
best fitness = sys.maxsize
best index = -1
best ind = np.zeros(( dimensions ))
worst fitness = 0
worst index = -1
worst ind = np.zeros(( dimensions ))
fit matrix = np.zeros(( pop size ))
for i in range( pop size ):
fitness value = fitness( population[i], dimensions, f )
fit matrix[i] = fitness value
if fitness value ¡ best fitness:
best fitness = fitness value
best index = i
best ind = population[i]
if fitness value ¿ worst fitness:
worst fitness = fitness value
worst index = i
```

```
worst ind = population[i]
for g in range( gens):
if g == 0 or g==gens-1:
print( best fitness)
print(population)
print()
for i in range(pop size):
new value = np.zeros(( dimensions ))
for j in range(dimensions):
new value[j] = population[i][j] + random()*( best ind[j] -
abs(population[i][j]) ) - random()*( worst ind[j] -
abs(population[i][j]))
new value[j] = population[i][j] + random()*( best ind[j] - worst ind[j])
new value fit = fitness( new value, dimensions, f )
current fit = fit matrix[i]
if new value fit ¡ current fit:
population[i] = new value
fit matrix[i] = new value fit
for i in range( pop size ):
if fit matrix[i] ¡ best fitness :
best fitness = fit matrix[i]
best fitness index = i
best ind = population[i]
if fit matrix[i] ¿ worst fitness :
worst fitness = fit matrix[i]
worst fitness index = i
best ind = population[i]
```

```
print(best fitness)
return best fitness
l = [ 1, 2, 3, 4, 5, 6, 7]
for i in l: print(f)
n = 20
ans = 0
for j in range(n):
ans += RAo(i)
ans = ans/n
if i == 1:
print( 'ackley ', ans)
if i == 2:
print( 'bealey ', ans)
if i==3:
print( 'booth ',ans)
if i==4:
print( 'easom ', ans)
if i==5:
print( 'matyas ', ans)
if i==6:
print( 'rastring ', ans)
if i==7:
print( 'rosenbrock ', ans)
# check for every case
```

5. Rao Application

5.1 Multi-Demand Multidimensional Knapsack problem

Maximize $\Sigma^{n}_{i=1}$ PiXi

Subject to $\Sigma^{n}_{i=1}$ WikXi$\leq$ C^{k} k € {1..... m }

Σ i=1 WikXi $\geq C^{k}$ k € {m + 1 .....m + Q}

Xi € {1...0} i € {1,...,n}

It is an extension of the classical knapsack hassle wherein knapsack has a hard and vast dimension. Each size is referred to as a knapsack constraint. With demand that are well known less-than-or-equal-to inequalities greater-than-or-equal-to inequalities. it could be described with the aid of a set of N items and a knapsack with m dimensions. The knapsack has a limited capacity in each size, okay denoted by using ck. Each object j has an income pj and a weight in every size, denoted by means of wk,j . The intention is to pick a subset of items with most total earnings, Chosen items must, however, not exceed knapsack constraints, The 0-1 decision variables xj suggest which gadgets are selected.

5.2 Example for understanding

It working on population size 8 with 2 knapsack constraint (m) 2 demand constraint

Input:

$N \leftarrow 8 \;\; m \leftarrow 2 \;\; q \leftarrow 2$

$\rightarrow Profits \rightarrow$

p_j	10	20	30	40	50	60	70	80

$\rightarrow Weights \rightarrow$

w_j^k	5	20	25	35	40	45	55	60
	90	120	70	110	90	65	80	150
	5	20	100	35	60	45	50	60
	90	60	70	110	90	45	20	10

$\rightarrow Capacity \rightarrow$

c^k	150	300	80	200

Constraints:

$$5x_1 + 20x_2 + 25x_3 + 35x_4 + 40x_5 + 45x_6 + 55x_7 + 60x_8 \leq 150$$
$$90x_1 + 120x_2 + 70x_3 + 110x_4 + 90x_5 + 65x_6 + 80x_7 + 150x_8 \leq 300$$
$$5x_1 + 20x_2 + 100x_3 + 35x_4 + 60x_5 + 45x_6 + 50x_7 + 60x_8 \geq 80$$
$$90x_1 + 60x_2 + 70x_3 + 110x_4 + 90x_5 + 45x_6 + 20x_7 + 10x_8 \geq 200$$
$$x_j \in \{0, 1\} \;\; (j = 1, \ldots, 8)$$

Objective:

$$maximize \;\; 10x_1 + 20x_2 + 30x_3 + 40x_4 + 50x_5 + 60x_6 + 70x_7 + 80x_8$$

Optimal solution:

x_j	0	0	0	1	1	0	1	0	value	160

5.3 Datasets Standardized

There are 9 input files:input_mdmkp c t1, ...,input_ mdmkp c t9.

These records files each contain ninety test issues which are the test problems from "A nearby search based heuristic for the demand restricted multidimensional knapsack problem", INFORMS JOC, 2003, to appear.

Give problem is extension of Multidimensional Knapsack problem with greater-than-inequality & same old less-than-identical conditions So we call it Multi-Demand Multidimensional Knapsack Problem

As we define problem equation (5.1)

Given a MKP occasion with m = limitations, 6 MDMKP examples are produced: one for each combination of costs type (either positive or blended) and number q of = requirements (q = 1, q = m/2 and q = m individually)

The format of the data file is:

number of test problems (K=15)

then for each test problem k (k=1,...,K) in turn:

number of variables (n), number of ¡= constraints (m)

for each <= constraint i (i=1,...,m): the coefficients a(i,j) j=1,...,n

for each <=constraint i (i=1,...,m): the right-hand side b(i)0

for each >= constraint i (i=m+1,...,m+m): the coefficients a(i,j) j=1,...,n

for each >= constraint i (i=m+1,...,m+m): the right-hand side b(i)

6 cost coefficients vectors c(j) j=1,...,n

(the first 3 are positive cost cases for q=1, q=m/2 and q=m+5 constraints ; the last 3 ones mixed cost case for q=1, q=m/2 and q=m constraints respectively).

Example with 6 cases

(out of 15)the first one in the file input _mdmkp c t1.txt(n = 100 and m = 5)

100 5 (number of variables (n), number of constraints (m))

42 41 523 ... 946 651 298 (weights 1 dime)

509 883 229 ... 805 881 850

806 361 199 ... 331 254 447

404 197 817 ... 993 525 322

475 36 287 ... 261 350 635

11927 13727 11551 13056 13460 (capacities)

556 125 736 ... 223 981 143

413 794 494 ... 945 272 849

907 447 158 ... 738 733 891

547 399 669 ... 630 91 525

599 63 627 ... 107 658 636

13162 13388 12431 12309 10949 (capacities)

835 1102 631 ... 1512 526 1380 (Profit vectors)

980 792 860 ... 774 752 1047

322 797 433 ... 839 591 341

248 494 15 ... 859 -266 646

-86 289 276 ... 227 -21 418

-191 -237 -183 ... 510 -128 -69

5.4 Implementation

We have been operating to create a running implementation of a binary derivation of the Rao algorithm which attempts to clear up optimization issues through generations of populations that trade over time to technique the current excellent and move far away from the present day worst inside the preceding population. To take a look at this implementation, it was carried out to a famous set of MDMKPs (Multiple Demand Multidimensional Knapsack Problems) from the net which then had to be changed to MDMMKPs (Multiple Demand Multidimensional Multiple choice Knapsack Problems).

Our first order of commercial enterprise was to copy this hassle set from the net, interpret the statistics, and write a program that could convert the problems into facts that might be manipulated through the algorithm. Once we had completed that, we needed to logically apply the selection constraints over the trouble set. Dr. Vasko,Dr. Lu and we had agreed that the selection constraints could wreck the variables in every hassle into instructions of 10 variables each, beginning with the first 10 variables from left to proper in the hassle description being assigned to first magnificence, the next 10 assigned to 2d elegance, etc.

Upon applying the selection constraints to the hassle set, it has become quickly obvious that the resulting troubles have been infeasible with the aid of their very nature. After trying out those issues with each Cplex and my very own code, we decided we needed to regulate the issues over again for us to even have feasible solutions to find. Given that we are restricting our maximum quantity of viable variables to use in our answers by way of 1/10th of the unique choice by forcing each 10 variables into a class, I cautioned that we try reducing right sides of the demand constraint all the way down to 1/tenth of their unique values. Doing this allowed Cplex to find possible solutions.

Now that the problems we have been using have had possible solutions, We started putting collectively the code necessary to implement the Rao algorithm at the MDMMKPs we

had created. For each problem, my program could randomly generate a preliminary set of 200 solutions with the aid of randomly deciding on a unameed variable in every magnificence to identical 1 and the relaxation equal zero. The random wide variety generator is seeded at the beginning of the program, ensuring that running the program multiple times will yield the identical results. It then types them first by violations in descending order, then by way of objective feature cost in ascending order, then takes the top 30 for our first population. Before each new release of Rao to generate the next population, We took the exceptional and worst solutions primarily based on the above criteria and shop references to them for the Rao algorithm's components for creating the following era of answers. The system expressed inside the original paper is supposed for continuous troubles.

For the MDMMKP, which has binary answers, we had to trade the definitions of the method.

The random numbers R1 and R2 have been changed from being factors of the actual numbers between 0 and 1 to handiest being elements of the set zero, 1. Next, while determining the new fee of a variable, the components has the capability to go back numbers aside from 0 or 1. In order to make this make sense within the context of MDMMKP, we had to convert these non-binary values to binary values. To do this, we agreed that if the returned end result is much less than or equal to 0, then we would update the formulation cost with 0. Otherwise,we would update it with a 1.

To make certain that class constraints aren't violated, after each new answer, We rent a greedy algorithm on it We talk to as my classify algorithm. The set of rules checks to look what number of variables are set to one within a class. If it's far zero, then it makes the variable with the highest corresponding objective feature coefficient equal 1. If it's far 1, then do nothing. If it's far more than 1, then most of the variables which might be currently assigned 1, whichever has the best corresponding objective characteristic coefficient is about 1 and the relaxation is set to 0. Due to using this function for ensuring class constraints are obeyed, any violation of sophistication constraints is not added to the violations rating for answers.

At this point inside the code, We have 2 implementations for creating the next era. The first implementation, which we can seek advice from as ordinary Rao, tests to look if the newly generated answer is higher than its corresponding answer from the preceding generation. If the new solution has much less violations, then replace the preceding answer. If the new answer has the equal quantity of violations and a higher objective characteristic, then update the previous answer. Otherwise, preserve the preceding solution.

Next inside the implementation, We delivered early termination standards to the Rao iterations. In this instance, Rao returns the nice end result if the nice answer does not now improve within 10 iterations with a maximum quantity of iterations set at 200.

In addition to this, We have blocks of code that run a community seek at the excellent solution of the final generation and after every new release. By commenting those blocks out or not now, we have been able to check the usefulness of the neighborhood seek at distinctive points within the code. The community seek .WE use tests one elegance at a time, sorts the variables in descending order by means of their corresponding objective function values, and if the best wasn't 1 to begin with, make it 1, change the rest to 0, then if violations move down or live the identical and the objective function is going up, then update the old answer with this new solution and go to the following class.

Otherwise, check the variable with the next highest objective function coefficient and attempt to make it 1 if it isn't always already. Continue till the next variable is already 1 or the next variable is swapped to one, then test the next class. Do this for each elegance once in a left to right analyzing through the variables. After the neighborhood search is complete, I run the classify set of rules on the new solution to again put in force the class constraints.

After this system has finished walking on a unnamed hassle, it adds the answer, the objective characteristic price, and the wide variety of iterations run to an excel spreadsheet. Each of these sheets has the list of indices of variables that the answer set to one, leaving the rest at 0, for 6 issues whose constraints all come from the identical set of constraints. All of the sheets for problems inside one data set are saved inside the equal book.

5.5 Output comparison:

NoofVariables	Noofconstraints	NoofClasses	case01	case02	case03	case04	case05	case06	Row averages
100	5	5	13.10	15.78	9.63	14.26	14.21	8.04	12.51
250	5	13	15.30	16.25	13.27	19.68	20.62	12.15	16.26
500	5	25	24.97	23.19	17.56	30.11	22.39	18.61	22.80
100	10	5	22.33	17.60	15.05	26.00	20.36	13.46	19.13
250	10	13	26.58	18.99	16.35	27.49	17.51	15.02	20.32
500	10	25	34.02	25.66	29.29	30.02	22.50	29.01	28.42
100	30	5	25.96	17.15	21.86	21.22	21.36	20.78	21.39
250	30	13	25.20	26.70	29.98	25.04	28.65	27.36	27.16
500	30	25	36.91	26.17	39.20	25.72	34.28	32.33	32.44
Column averages			24.93	20.86	21.35	24.39	22.43	19.64	22.27

These are the standard result taken from cplex (reference paper of lu2019)

6. Results and Discussion

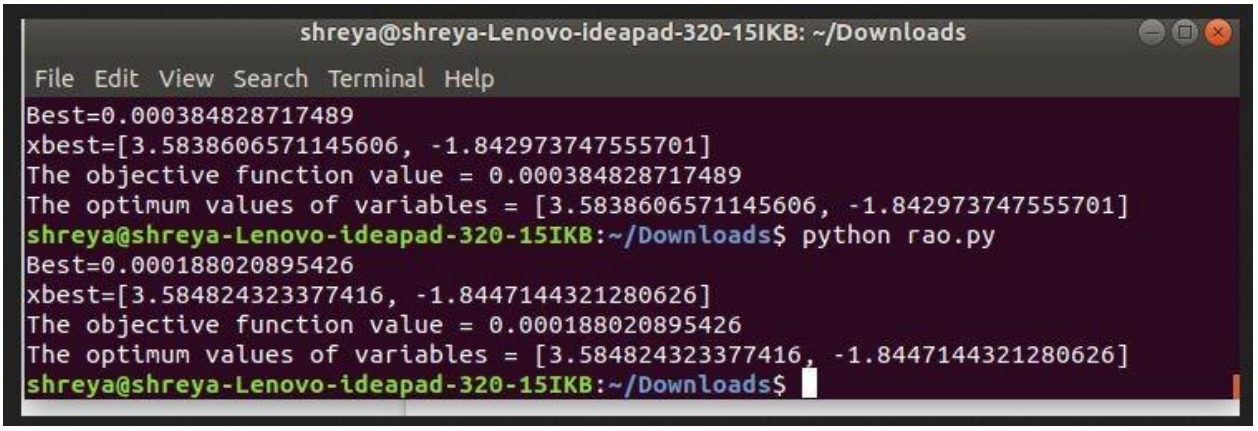

Fig 4: Single pc execution result of Rao Algorithm .

```
 9     0.0644   -0.0185    4.4958
10
11 Iteration No. = 2
12 %%%%%%%% Final population %%%%%%%%%
13    1.0e+03 *
14
15    -0.0257   -0.0141    0.8617
16    -0.0110    0.0258    0.7871
17     0.0593    0.0081    3.57924
18
19    -0.0410    0.0362    2.9858
20     0.0644   -0.0185    4.4958
21
22 Iteration No. = 3
23 %%%%%%%% Final population %%%%%%%%%
24    1.0e+03 *
25
26    -0.0257   -0.0141    0.8617
27    -0.0110    0.0258    0.7871
28     0.0216    0.0409    2.1413
29    -0.0410    0.0362    2.9858
30     0.0055   -0.0031    0.0393
31
32 Iteration No. = 4
33 %%%%%%%% Final population %%%%%%%%%
34    1.0e+03 *
35
36    -0.0257   -0.0141    0.8617
37    -0.0069    0.0086    0.1206
38     0.0326    0.0172    1.3595
39     0.0014    0.0073    0.0554
40     0.0055   -0.0031    0.0393
41
42 Iteration No. = 5
43 %%%%%%%% Final population %%%%%%%%%
44   -25.7218  -14.1446  861.6818
45    -6.8612    8.5721  120.5564
46    12.9836   10.0536  269.6482
47     1.3769    7.3170   55.4342
48     1.5261   -5.7080   34.9100
49
50 Iteration No. = 6
51 %%%%%%%% Final population %%%%%%%%%
52   -24.6724   -6.7265  653.9756
53    -6.8612    8.5721  120.5564
54    12.9836   10.0536  269.6482
55     1.3769    7.3170   55.4342
56     1.7664   -4.0979   19.9133
57
58 Optimum value = 19.91328176
```

Fig : Single pc execution result of Rao Algorithm with 5 Iteration.

```
The objective function value = 0.0006235463964256088
The optimum values of variables = [3.5878826255085396, -1.8482237730479525]
```

Fig 6: BenchMark Function Rao's algorithm

```
        print( 'rastring   ', ans)
    if i==7:
        print( 'rosenbrock  ', ans)
```

```
ackley      1.1722266131898695
bealey      0.378043569406309
booth       1.0189905127176186
easom       -0.0380310946429861
matyas      0.028816147429688306
rastring     2.594765487968942
rosenbrock   2.3651639050585324e+18
```

Fig 7: BenchMark Function Rao's algorithm

```
Columns 1 through 18

  -0.0111   -0.0072    0.0184   -0.0094   -0.0111    0.0043    0.0013    0.0117   -0.0254    0.0056   -0.0012   -0.0008    0.0068    0.0066   -0.0236   -0.0002    0.0026    0.0333
  -0.0138   -0.0055    0.0187   -0.0032   -0.0041    0.0050    0.0026    0.0165   -0.0265    0.0043    0.0052   -0.0097    0.0062    0.0076   -0.0247    0.0008    0.0026    0.0310
  -0.0111   -0.0051    0.0192   -0.0064   -0.0147    0.0045    0.0053    0.0109   -0.0263    0.0031    0.0033   -0.0051    0.0061    0.0130   -0.0227   -0.0015    0.0027    0.0325
  -0.0108   -0.0072    0.0194   -0.0094   -0.0094    0.0042    0.0008    0.0122   -0.0250    0.0037    0.0025   -0.0013    0.0039    0.0096   -0.0239   -0.0008    0.0026    0.0326
  -0.0110   -0.0066    0.0189   -0.0025   -0.0132    0.0052    0.0008    0.0072   -0.0265    0.0001    0.0013   -0.0071    0.0052    0.0169   -0.0234    0.0003    0.0026    0.0321
  -0.0104   -0.0073    0.0201   -0.0105   -0.0071    0.0037    0.0023    0.0112   -0.0254   -0.0002    0.0118   -0.0014    0.0040    0.0134   -0.0251   -0.0095    0.0022    0.0304
  -0.0111   -0.0056    0.0194    0.0015   -0.0135    0.0038    0.0022    0.0113   -0.0255    0.0029    0.0045   -0.0043    0.0068    0.0111   -0.0244   -0.0066    0.0027    0.0313
  -0.0111   -0.0043    0.0174   -0.0126   -0.0116    0.0042    0.0024    0.0071   -0.0248    0.0053    0.0097   -0.0119    0.0072    0.0115   -0.0251   -0.0041    0.0027    0.0310
  -0.0110   -0.0034    0.0176   -0.0079   -0.0116    0.0041    0.0025    0.0043   -0.0248    0.0053    0.0095   -0.0122    0.0073    0.0115   -0.0252   -0.0041    0.0027    0.0308
  -0.0109   -0.0001    0.0179   -0.0050   -0.0112    0.0020   -0.0035    0.0002   -0.0272    0.0082    0.0006    0.0057    0.0064    0.0118   -0.0261   -0.0109    0.0028    0.0315

Columns 19 through 21

   0.0053   -0.0241    3.9369
   0.0040   -0.0237    3.9906
   0.0046   -0.0246    4.0961
   0.0059   -0.0241    3.8974
   0.0099   -0.0256    4.1819
   0.0079   -0.0248    4.1822
   0.0125   -0.0247    4.1221
   0.0079   -0.0256    4.2060
   0.0082   -0.0256    4.0805
   0.0128   -0.0254    4.2351


best=1759.798903
mean=2555.385902
worst=3897.392241
std. dev.=797.504266
mean Fes=496.000000
>>
```

Fig 8: Self-adaptive Multi-population Rao Algorithm 1

(considering with 10 runs and 500 function evaluation(value))

```
 Columns 1 through 18

    0.0000   -0.0044   -0.0003   -0.0003    0.0000   -0.0063    0.0022    0.0004   -0.0006   -0.0031    0.0029    0.0002   -0.0031   -0.0002    0.0024    0.0005    0.
    0.0009   -0.0020    0.0005   -0.0004   -0.0003   -0.0002    0.0020   -0.0018   -0.0008   -0.0009    0.0061   -0.0003    0.0013    0.0023    0.0031   -0.0016    0.
    0.0001   -0.0018   -0.0001   -0.0009   -0.0010   -0.0008    0.0020    0.0000   -0.0005   -0.0018    0.0037   -0.0008   -0.0041   -0.0003    0.0038   -0.0016    0.
   -0.0004   -0.0015   -0.0008   -0.0005   -0.0017   -0.0019    0.0033    0.0002   -0.0005    0.0004    0.0039   -0.0024   -0.0072   -0.0003    0.0022   -0.0015    0.
    0.0028    0.0012   -0.0026   -0.0006   -0.0053   -0.0024    0.0022    0.0003   -0.0001   -0.0018    0.0043    0.0000   -0.0035   -0.0035    0.0007   -0.0002    0.
    0.0009   -0.0014   -0.0011   -0.0003   -0.0003   -0.0011    0.0034    0.0009    0.0004    0.0010    0.0016   -0.0034   -0.0023   -0.0000    0.0031   -0.0011    0.
   -0.0015   -0.0021    0.0006    0.0015   -0.0028   -0.0016    0.0049    0.0005   -0.0001   -0.0023    0.0030   -0.0019   -0.0028   -0.0012   -0.0003   -0.0011    0.
   -0.0011   -0.0025   -0.0000    0.0008    0.0017   -0.0006    0.0043   -0.0003    0.0001   -0.0012    0.0020   -0.0032   -0.0018   -0.0051    0.0003   -0.0022    0.
   -0.0004   -0.0018   -0.0003    0.0011   -0.0029   -0.0022    0.0036    0.0008   -0.0005   -0.0056    0.0042   -0.0018   -0.0017    0.0001    0.0001    0.0006    0.
   -0.0031   -0.0018    0.0019   -0.0020   -0.0057    0.0022    0.0030    0.0007   -0.0012   -0.0020   -0.0000   -0.0006    0.0000    0.0019    0.0013   -0.0000    0.

 Columns 19 through 21

   -0.0026    0.0025    1.2339
   -0.0035   -0.0023    0.9896
   -0.0036    0.0019    0.9147
    0.0024   -0.0016    1.2118
   -0.0019    0.0010    1.1739
   -0.0034    0.0000    0.6870
   -0.0048   -0.0008    1.0700
   -0.0038    0.0004    1.0773
   -0.0039    0.0001    1.1525
   -0.0039   -0.0024    1.0594


best=4242.122822
mean=8022.446173
worst=13078.687399
std. dev.=2742.198304
mean Fes=439.000000
>>
```

Fig 9: Self-adaptive Multi-population Rao Algorithm 2

(considering with 10 runs and 500 function evaluation(value))

```
Columns 1 through 18

   0.0094    0.0110    0.0008    0.0039   -0.0002    0.0016   -0.0052    0.0103    0.0109   -0.0036   -0.0049    0.0176   -0.0001    0.0137    0.0013    0.0033    0.0062   -0.0019
   0.0133    0.0097    0.0046    0.0069   -0.0002    0.0032   -0.0042    0.0054    0.0115    0.0007   -0.0035    0.0138   -0.0050    0.0161    0.0015    0.0064    0.0074    0.0027
   0.0100    0.0126    0.0025    0.0056   -0.0002    0.0026   -0.0097    0.0108    0.0104   -0.0037    0.0035    0.0167   -0.0024    0.0178    0.0016    0.0003    0.0081    0.0027
   0.0076    0.0139    0.0012    0.0043   -0.0002    0.0033   -0.0013    0.0090    0.0100   -0.0110    0.0035    0.0184    0.0013    0.0166    0.0015    0.0018    0.0087    0.0100
   0.0100    0.0118    0.0052    0.0025    0.0000    0.0068    0.0085    0.0122    0.0105   -0.0024   -0.0065    0.0162    0.0005    0.0167    0.0015    0.0042    0.0098    0.0028
   0.0072    0.0126    0.0037    0.0075    0.0008    0.0077   -0.0018    0.0102    0.0107   -0.0061    0.0018    0.0148   -0.0047    0.0079    0.0013    0.0021    0.0046    0.0051
   0.0093    0.0111    0.0025    0.0066    0.0001    0.0111   -0.0062    0.0155    0.0103    0.0041   -0.0001    0.0190   -0.0001    0.0109    0.0002    0.0071    0.0056    0.0079
  -0.0020    0.0134    0.0043    0.0120    0.0014    0.0107   -0.0056    0.0050    0.0102   -0.0034    0.0093    0.0171   -0.0050    0.0003   -0.0006   -0.0035    0.0113    0.0093
   0.0007    0.0133    0.0031    0.0073    0.0005    0.0113   -0.0068    0.0113    0.0100   -0.0077    0.0075    0.0161   -0.0032    0.0092    0.0015    0.0024    0.0083    0.0014
   0.0023    0.0131    0.0041    0.0115    0.0008    0.0140   -0.0046    0.0086    0.0101   -0.0085   -0.0011    0.0151    0.0013    0.0046    0.0010   -0.0054    0.0061    0.0074

Columns 19 through 21

  -0.0053    0.0168    1.3770
   0.0002    0.0199    1.5179
  -0.0013    0.0161    1.5840
  -0.0019    0.0164    1.6676
  -0.0019    0.0162    1.6273
  -0.0012    0.0144    1.1695
  -0.0017    0.0169    1.6975
  -0.0069    0.0137    1.4961
   0.0019    0.0090    1.2749
   0.0027    0.0165    1.4430


best=918.656663
mean=1960.306625
worst=3933.624222
std. dev.=1181.054523
mean Fes=473.000000
>>
```

Fig 10: Self-adaptive Multi-population Rao Algorithm 3

(considering with 10 runs and 500 function evaluation(value))

A	B	C	D	E	F	G	H	I	J	K	L	M
13981	14	12398	28	10302	44	7383	55	6690	66	5818	77	
11968	16	12041	32	11171	48	7496	60	6522	71	6272	82	
13295	13	10672	27	9225	38	6993	49	6241	60	5833	71	
12790	13	12073	28	9993	41	7357	52	6304	63	6067	74	
14328	13	12516	28	11030	46	7167	57	6546	68	6205	79	
12857	12	11169	27	9963	39	6852	51	6166	64	5846	75	
13224	13	11929	26	10017	39	6298	50	5957	63	6121	74	
13772	11	11923	24	9625	36	6570	48	5999	60	5771	71	
13081	11	12055	26	9347	41	7095	55	6306	66	5851	77	
12530	11	11946	28	9705	42	7297	57	6493	68	6156	79	
12345	11	11362	23	9478	35	5773	47	4895	58	5334	70	
12365	12	10512	24	9214	36	5574	48	6038	59	5472	72	
11998	11	11383	24	9917	36	5628	48	5637	60	6036	73	
13516	11	11175	24	11368	36	6075	47	5581	59	6417	72	
11413	11	10535	23	8772	34	5533	46	5638	58	5295	70	

Fig 11:knapsack with cp1 input file Result.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
input filename 1 : mdmkp_ct1.txt.txt				MDMKP_Case														
	MDMKP_Cas	MDMKP_Cas	MDMKP_Cas	MDMKP_Cas	MDMKP_Cas	Bits1	Opt_Sol2	UT2	Bits2	Opt_Sol3	UT3	Bits3	Opt_Sol4	UT4	Bits4	Opt_Sol5	UT5	Bits5
1	1	100	5	13981	0.008	10	12585	0.007	10	10302	0.009	10	7383	0.008	10	6690	0.009	10
1	2	100	5	12017	0.008	10	12041	0.008	10	11456	0.009	10	7496	0.008	10	6522	0.009	10
1	3	100	5	13295	0.008	10	10789	0.008	10	9225	0.009	10	6993	0.009	10	6241	0.008	10
1	4	100	5	12927	0.007	10	12643	0.008	10	9993	0.009	10	7357	0.008	10	6304	0.008	10
1	5	100	5	14328	0.008	10	13000	0.008	10	11030	0.009	10	7167	0.007	10	6546	0.007	10
1	6	100	5	13196	0.019	10	11993	0.008	10	10444	0.009	10	7291	0.02	10	6212	0.016	10
1	7	100	5	13647	0.009	10	12873	0.008	10	10397	0.009	10	6761	0.021	10	6313	0.02	10
1	8	100	5	14187	0.018	10	11923	0.008	10	9692	0.011	10	7037	0.014	10	6180	0.015	10
1	9	100	5	13426	0.009	10	12629	0.014	9	9761	0.009	10	7387	0.01	10	6306	0.008	10
1	10	100	5	13581	0.014	10	12710	0.016	10	9934	0.009	10	7297	0.009	10	6493	0.015	10
1	11	100	5	12959	0.022	10	11707	0.009	10	9756	0.028	10	6319	0.047	10	5833	0.022	10
1	12	100	5	13090	0.019	10	10936	0.027	10	9663	0.03	10	5991	0.034	10	6442	0.026	10
1	13	100	5	12606	0.014	10	12410	0.034	10	10676	0.019	10	6246	0.018	10	6118	0.032	10
1	14	100	5	13803	0.022	10	12163	0.036	10	12175	0.043	10	6473	0.041	10	6226	0.092	10
1	15	100	5	12115	0.019	10	11451	0.023	10	9652	0.017	10	6214	0.01	10	5885	0.03	10
input filename 2 : mdmkp_ct2.txt.txt																		
MDMKP_Cas	Prob_ID	Num_var	Num_const	Opt_Sol1	UT1	Bits1	Opt_Sol2	UT2	Bits2	Opt_Sol3	UT3	Bits3	Opt_Sol4	UT4	Bits4	Opt_Sol5	UT5	Bits5
2	1	250	5	37168	0.011	25	33416	0.012	25	28588	0.016	25	18402	0.011	25	16585	0.013	25
2	2	250	5	34112	0.011	25	32827	0.013	25	27167	0.016	25	18805	0.012	25	16766	0.013	25
2	3	250	5	35002	0.013	25	30957	0.012	25	27676	0.016	25	18557	0.012	25	15858	0.012	25
2	4	250	5	35524	0.011	25	33755	0.012	25	29035	0.016	25	18848	0.012	25	16702	0.012	25
2	5	250	5	35713	0.012	25	31074	0.013	25	26508	0.016	25	19846	0.012	25	16459	0.013	25
2	6	250	5	37264	0.017	25	32927	0.013	25	30812	0.024	25	18276	0.023	25	16586	0.012	25
2	7	250	5	33685	0.011	25	31957	0.012	25	28921	0.017	25	18289	0.046	25	15965	0.024	25
2	8	250	5	35095	0.027	25	32158	0.014	25	25980	0.017	25	17357	0.155	25	16334	0.024	25
2	9	250	5	34710	0.017	25	29453	0.018	25	26395	0.016	25	17853	0.167	25	16782	0.026	25
2	10	250	5	33791	0.019	25	31904	0.015	25	28784	0.016	25	18723	0.059	25	16305	0.014	25
2	11	250	5	36190	0.058	25	32698	0.068	25	29195	0.063	25	16584	0.012	25	15556	0.067	25
2	12	250	5	34470	0.02	25	30397	0.056	25	30387	0.054	25	16359	0.229	25	16145	0.035	25
2	13	250	5	36905	0.023	25	32669	0.066	25	27378	0.076	25	16529	0.076	25	16348	0.081	25
2	14	250	5	33148	0.024	25	31040	0.052	25	27025	0.067	25	16174	0.09	25	16334	0.044	25
2	15	250	5	33869	0.026	25	32273	0.026	25	27251	0.032	25	16067	0.12	25	15864	0.025	25

Fig12 :knapsack with 10 input's files Result

6.2 Summary of performance study

Numerical Solutions of the Rao & SAMP Algorithms over considering 30 runs and considered (500,000 function evaluations(value))

Function	Optimum	Rao1	Rao2	Rao3	SAMP-Rao1	SAMP-Rao2	SAMP-Rao3
	0 b	0	0	0	0	0	0
	w	0	0	0	0	0	0
F1	m	0	0	0	0	0	0
	sd	0	0	0	0	0	0
	mef	499876	499781	276522	49896	498760	263470
	0 b	0	0	0	0	0	0
	w	0	0	0	0	0	0
F2	m	0	0	0	0	0	0
	sd	0	0	0	0	0	0
	mef	499875	499751	27656	496893	498831	275914
	0 b	0	0	0	0	0	0
	w	0	0	0	0	0	0
F3	m	0	0	0	0	0	0
	sd	0	0	0	0	0	0
	mef	9806	7613	7425	4796	3791	5868
	-1 b	-1	-1	-1	-1	-1	-1
	w	0	-1	-1	-1	-1	-1
F4	m	-0.5687	-1	-1	-1	-1	-1
	sd	0.5060	0	0	0	0	0
	mef	3040	11887	14035	52266	3210	6527
	0 b	0	0	0	0	0	0
	m	0	0	0	0	0	0
F5	m	0	0	0	0	0	0

	sd	0	0	0	0	0	0
	mef	77023	11054	14308	2766	41647	55044
	0 b	0	0	0	0	0	0
	w	0	5.35-23	1.32	0	0	0
F6	m	0	1.80	7.87	0	0	0
	sd	0	9.6-24	2.61	0	0	0
	mef	38506	47753	488127	130478	156904	155109
	-50 b	-50	-50	-50	-50	-50	-50
	w	-50	-50	-50	-50	-50	-50
F7	m	-50	-50	-50	-50	-50	-50
	sd	0	0	0	0	0	0
	mef	17486	37208	34795	16723	32212	32933
	-210 b	-210	-210	-210	-210	-210	-210
	w	-210	1171	-210	-210	-210	-210
F8	m	-210	-30.858	-210	-210	-210	-210
	sd	0	4.13	0	0	0	0
	mef	48231	144156	142253	42937	149932	136764
	0 b	0	0	0	0	0	0
	w	0	0	0	0	0	0
F9	m	0	0	0	0	0	0
	sd	0	0	0	0	0	0
	mef	3445615	499766	258451	283001	499706	256044
	0 b	0	0	0	0	0	0
	w	0	0	0	0	0	0
F10	m	0	0	0	0	0	0
	sd	0	0	0	0	0	0
	mef	5582	4485	4312	2136	1889	1741

b: best outcome.

w: worst outcome.

m:mean result.

sd: standard deviation result.

mfe: mean function evaluations result**.**

7 Conclusions and Future Work

Through our experiments, we've understood the concept of Rao's algorithm , we demonstrate the running of Rao's Algorithm using a population base dataset where we compare the result of Sequential and Spark Based Model via applying some Benchmark Function on it. To check it's functionality. future work to take any optimization trouble and solve using our spark based totally version Application with speed and efficiently.

References

[1] J. Dean, and S. Ghemawat, "MapReduce: simplified data processing on large clusters.The L A TEX Communications of the ACM,. vol. 51.1, pp. 107-113, 2008.

[2] R. V. Rao R. B. Pawar, "Self-adaptive Multi-population Rao Algorithms for Engineering Design Optimization". pp. 65, 22 Jan 2020.

[3] R. V Rao,"Rao algorithm:three metaphor simple algorithm for solving optimization problem" of IJIECvol 11,2020

[4] Deb, and Kalyanmoy, "Optimization for engineering design algorithms and examples," ˜PHI Learning Pvt. Ltd., 2012.

[5] P.C. Chu and J.E. Beasley "A genetic algorithm for the multidimensional knapsack problem", Journal of Heuristics, vol. 4, 1998, pp63-86.

[6] Beasley J. OR Library, 1995. Available at http://www.ms.ic.ac.uk/info.html. Demand constrained MDKP test cases are at http://mscmga.ms.ic.ac.uk/jeb/orlib/mdmkpinfo.html.

[7] Yun Lu Francis J. Vasko A comprehensive empirical demonstration of the impact of choice constraints on solving generalizations of the 0–1 knapsack problem using the integer programming option of CPLEX.