Comparative Analysis of Genetic Algorithm and Classical Algorithms in Fractional Programming

Debasish Roy, Surjya Sikha Das and Swarup Ghosh

Abstract This paper compares the performances of genetic algorithm with various classical algorithms in solving fractional programming. Genetic algorithm is one of the new forms of algorithms for solving optimization problems, which may not be efficient but a generic way to solve nonlinear optimization problems. The traditional optimization algorithms have difficulty in computing the derivatives and second order partial derivatives, i.e., Hessian for the fractional function. The issues of discontinuity seriously affects traditional algorithm. There are large numbers of classical methods for searching the optimum point of nonlinear functions. The classical search algorithms may be largely classified as gradient based methods and nongradient methods. Here, a comparative performance analysis of different algorithms is made through a newly defined function called algorithmic index. An algorithm based on heuristics for computation of gewicht vector required to derive algorithmic index has also been proposed here.

Keyword Fractional programming · Genetic algorithm · Optimization · DEA

1 Introduction

Rechenberg, a German scientist, introduced evolutionary strategies for airfoil shape point optimization, in the 1960s. Fogel, Owen, and Walsh formulated evolutionary programming for finite state machines. Evolutionary computation is a broad field

D. Roy (✉) · S.S. Das
Department of Management Studies, Techno India University, Kolkata, India
e-mail: debasishroy7@gmail.com

S.S. Das
e-mail: surjyasikha.tiu@gmail.com

S. Ghosh
Department of Humanities and Social Sciences, Techno India University, Kolkata, India
e-mail: swarupghosh55.tiu@gmail.com

© Springer India 2016
R. Chaki et al. (eds.), *Advanced Computing and Systems for Security*,
Advances in Intelligent Systems and Computing 396,
DOI 10.1007/978-81-322-2653-6_17

and remains a serious interest for research with sub areas as evolutionary programming, evolutionary strategies, and genetic algorithm. Large number of researchers worked on evolution-based algorithms, notable among them are Box (1957); Friedman (1959); Bledsoe (1961); Bremermann (1962); Reed, Tombs, and Baricelli (1967).

Genetic algorithm, developed by John Holland who published a book titled 'Adaptation in Natural and Artificial Systems' in 1960 in the University of Michigan, mimic adaptation and traverses from one set of population to a new set of population by crossover, mutation and selection to achieve and find fitter, and more suitable population. Later schemas formed basis for all subsequent developments. This paper compares performances of traditional methods like Random Search, Box Evaluation, Gradient Descent method, and Hookes' Jeeves method with Genetic Algorithm on the Fractional Programming. In order to have an even comparison, same fractional function is taken for experimentation and optimized values have been derived using Matlab scripts. This paper also proposes algorithmic index for comparison of performances of these algorithms. The computation of algorithmic index depends on gewicht vector; estimation of this vector can be done by newly proposed heuristics dependent algorithm.

2 Literature Review

The optimization of the ratio of linear functions has attracted researchers for many years. In many practical applications, multiple such fractions need to be optimized. Von Neumann was the earliest to have started using fractional programming in Equilibrium problems, in 1937.The linear fractional programming started with B. Mortars and his associates. Charnes and Cooper [1] had proposed transformation LFP to Linear Programming format and thereby solving the problem. Bitranes and Novaes solved the problem by gradient descent method [2]. Bitranes and Magnant [3] made analysis in the same score as Linear Programming for duality and sensitivity. Swarup [4] extended simplex method to LFP, which is an extension of work of Danjtzig [5, 6], for solving LFP. Bounded variable linear fractional programming was explored by Bazalinov [7]. He also worked on scaling problems in LFP [8]. Interval Valued Fractional Programming was studied by Shohrab Effati [9]. Fractional Programming has often been addressed in the domain of generalized fractional programming [10].

Until 1980s, the single ratio problem has dominated the field. Various methods have evolved; important among them are use of the LPP methods for solving fractional functions. Gogia [11] suggested revised simplex method for solving fractional programming. Horvath [12] gave a criteria for determining optimum for linear fractional programming using duality considerations. A finite method for solution to a simpler form of the fractional programming was developed by Stahl [13]. A special class of linear fractional programming is fractional interval programming, which has been studied by a number of authors [14]. Buhler solved fractional interval programming using generalized inverses [15].

2.1 Random Search Method

Random search method uses a population created either in random manner or by performing unidirectional search along the random search direction. If the feasible space is narrow, the system of random search may fail. Random Search method has various algorithms starting from Pure Random Search and Random Walk to Metaheuristic. Pure Random Search [16] is a stochastic search method of selecting random population based on normal or Gaussian distribution. Convergence can be proved in number of ways [17–19]. The procedure may guarantee convergence, but is not efficient as it may require large number of iterations depending on nature of the objective function, feasible zone, and dimension of variables. It may be enormously large. One such Random Search algorithm is presented in supplementary material.

2.2 Box's Evolutionary Method

Introduced by G.E.P Box in 1957, this evolutionary algorithm, centering on a point, selects best point among 2 N points evaluated along N dimensions. The next iteration starts from this point. The size of the hypercube is successively reduced with change of the position of the center. Here, required number of function evaluation increases exponentially with N. This is one of the strong drawbacks of the Box's Method. There are other versions of Box's Evolutionary Method like Random Evolutionary Operation (REVOP), Simplex EVOP, etc. The convergence of the algorithm depends on initial hypercube size, position, and reduction rate, ∂_i. This is basically a multi-dimensional systemic search on dimensions [20]. The algorithm is useful since it is a derivative free optimization method (DFO) [21–23]. Subsequently, further improvement was made by Wilson [24] and it was renamed as Response Surface Methodology (RSM). Whereas RSM is based on least square, DFO is based on direct function values or its interpolation. The hypercube is the focal point for searching optimum point. The Box's method was also opted in finding optimum solutions in industrial applications [25]. The algorithm, however, does not guarantee convergence to local or global optima. The algorithm is presented in supplementary material.

2.3 Hooke Jeeves' Method

In 1961, Hooke and Jeeves [26] conceived that direct search method is effective when the objective function is nondifferentiable or does not have derivative at all points in feasible region. In this method, each trial is compared with the previous best [27]. Therefore, direct search methods for unconstrained optimization works on relative rank of countable set function values whereas, Armijo-Goldstein-Wolfe condition for quasi-Newton line search algorithm requires a sufficient decrease in objective function. The algorithm is presented in supplementary material.

2.4 Gradient Descent Method

Here, one of the two popular gradient descent methods is presented. The direct search method eventually requires large number of steps or iterations to converge, whereas gradient-based methods are faster. However, convex optimization methods define subdifferentials for functions having discontinuity, which is defined as follows:

$$\partial f\left(\overset{=}{x}\right) = \left\{v \in R^n : f(y) - f\left(\overset{=}{x}\right) \geq \left\langle v, y - \overset{=}{x}\right\rangle, \forall y \in R^n\right\}. \tag{1}$$

In cases where the objective function is differentiable, the gradient methods [28] or derivative-based methods are useful and efficient, compared to direct search method. One of the oldest systems of finding multivariable optimum points is Newton's Method [29]. This method is extremely important as it is the simplest method and assures convergence [30]. Large numbers of improved algorithms have emerged from this algorithm with slight or minimum modification. One of the most accepted methods based on the Newton Method and that has been widely accepted is the Conjugate Direction Method [31]. A generalized powerful extension of this is the Spacer Step Theorem [32]. The algorithm for Gradient Descent Method is given in supplementary material.

2.5 Genetic Algorithm

The schema based on genetic algorithm was developed in 1968, with famous disposition of Schema Theorem [33]. Goldberg [34] developed building block hypothesis from Schema Theorem. The criticism of Schema Theorem developed in 1990 that the effect of noise and other stochastic effects distort proportionate selection [35, 36, 37].

The algorithmic flow is given in the supplementary material. The parameters of genetic algorithm are

- Cross Over Probability—It is 0 %, if the offspring population is an exact copy of the parent. It is 100 %, if all of the parent population is allowed to crossover.
- Mutation Probability—Mutation probability is zero, if no population is changed after cross over. It is 100 %, if the whole chromosome is changed. Mutation is necessary to prevent falling of the population in local optimum.
- Population Size: If population size is small, the search space will not be covered well. If search space is large, the algorithm becomes slow. Population size also determines the precision of the solution, i.e., the quality of the solution.

Besides choosing appropriate population size, the balance between selection operator and exploration operator introduced by crossover and mutation operator is also important. If selection operator uses too much selection pressure, then the population loses diversity. Genetic algorithm is used for NP-hard problems. It is more robust than conventional algorithm, i.e., the algorithm does not collapse in the presence of noise or change of inputs. Also, genetic algorithm is useful in searching n-dimensional surface or multimodal search space.

3 Linear Fractional Programming

Linear Fractional Programming has various forms. Let p, q, and s denote real valued functions which are defined over $C \in R^n$. Let us take

$$s(x) = \frac{p(x)}{q(x)}. \tag{2}$$

The function s is defined over $D = \{x \in C : s(x) \leq 1\}$ assuming $q(x) \neq 0$ for $x \in C$.

Single Ratio Fractional Programming may be defined as

$$\mathrm{Max}\{s(x) : x \in D\}. \tag{3}$$

In many practical applications, multiple ratios appear for evaluation. This is also referred to as max−min problem. The Generalized Fractional Programming may be defined as

$$\mathrm{Max}\{\min s_i(x) : \ x \in D\}, \ \text{where } s_i(x) = \frac{p_i}{q_i}, \text{where } i = 1, 2\ldots m \ \text{and} \ s_i > 0. \tag{4}$$

We call it as concave fractional programming, if numerator p_i is concave on D. The denominator q_i is convex function on D. It is further assumed that p_i is non-negative on D, if q_i is not affine. The objective function in general is not assumed to be concave. The objective function is assumed to be a ratio of convex and concave function. The fractional programs are, in general, assumed to be nonconcave programs. The central point of fractional programming is objective function and point of attraction is the ratio structure with a feasible region being a convex polyhedron.

Sometimes, functions in both numerator and denominator are affine functions. If D is a convex polyhedron, the problem is called Linear Fractional Program. The form of the function is as follows:

$$Max\left\{\frac{a^T x + \emptyset}{b^T x + \theta} : Ax \leq c, x \geq 0\right\}, \text{Where } a, b \ \in \ R^n, \emptyset, \theta \ R, A \ \in \ R^{mxn}, c \ \in \ R^m. \tag{5}$$

4 The Experiment

In this experiment, a standard two variable (0–1) fractional function is taken as follows:

$$\text{Maximize} f(x,y) = \frac{3x + 4y + 1}{5x + 7y + 5} \left\{ \forall (x,y) \in \mathcal{R}^2 : 5x + 7y + 5 \neq 0 \right\}. \tag{6}$$

During the experiment, the optimum value that is the maximum value of this sample fractional function is obtained. The goal is to maximize the fraction. This is treated as unconstrained optimization problem. There are large numbers of methodologies available for finding the optimum value of this NP-hard problem. Here, three nonderivative-based methods and one derivative-based method is chosen. Genetic algorithm is also used to find the optimum value. The comparison of the computation time, number of iterations, and optimum values of all the algorithms is found. The parameter sensitivity and other studies can also be conducted. These algorithms are primarily numerical methods; as a result, comparison of efficiency of algorithms analytically is difficult. The efficiency of the algorithms depend not only on parameters but also on type of function. The experiment has been conducted largely on same setting, which is the same function and same computer. The precision of output is also chosen same in all cases. The basic nature of the algorithms prevents complete identical context generation. As a result, some intrinsic differentiation remains.

4.1 Random Search

Here, 1000 random values are chosen between 0 and 1, and function value is evaluated at all these points. The maximum of the function value is chosen. The random numbers are chosen based on normal distribution. The random numbers are 2-dimensional. The Matlab code is given below (Fig. 1)

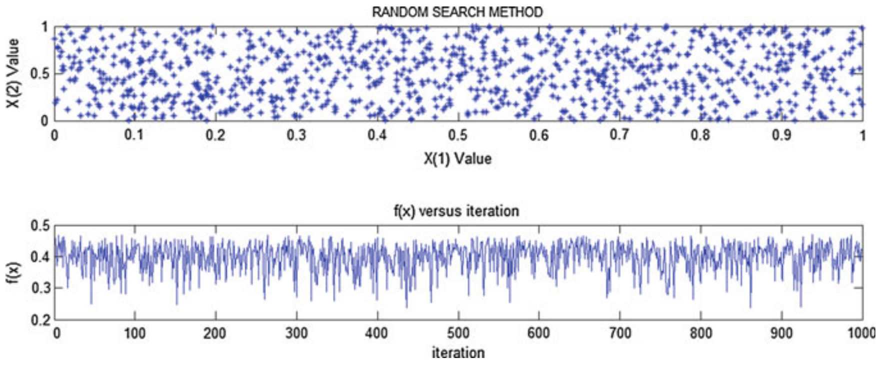

Fig. 1 The plots of iteration points and f(x,y) versus iteration

```
function
[foptxopttimespent]=rando
mOptim(iter)
tic;
xran=ones(iter,2);
f=ones(1,iter);
fori=1:iter
xran(i,:)=rand(1,2);
     X=xran(i,:);
f(i)=fx1(X);
holdon;
subplot(2,1,1);
The plot of output is:
plot(X(1),X(2),'*');
holdoff;
title('RANDOM SEARCH
METHOD');
end
[fopt p]=max(f);
xopt=xran(p,:);
subplot(2,1,2);
plot(f);
timespent=toc;
end
```

The output is as follows:

$$\text{fopt} = 0.4698, \text{xopt} = 0.99950.9810, \text{timespent} = 30.7039$$

4.2 *Box Evolutionary Method*

In the case of Box's Evolutionary method, initial point is chosen as (x,y) = (2,3). The precision or delta value is chosen as 0.01. The Matlab code is given below (Fig. 2)

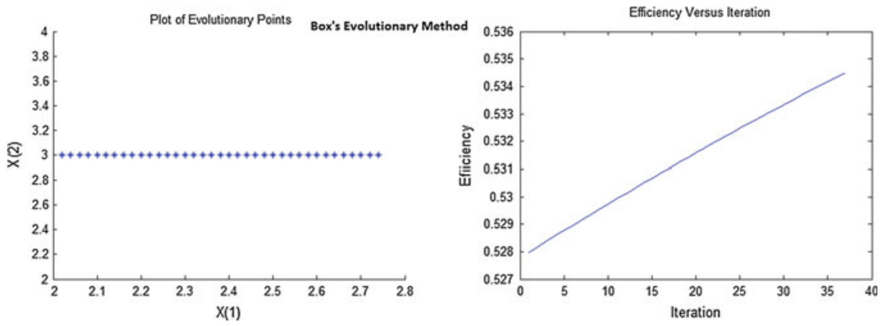

Fig. 2 The plot of (x,y) and f(x,y) with iterations

```
function
[fmaxXmaxiteriterTime]=bo
xEval(xinit,d)
clc;
tic;
iter=0;
x=xinit;
delta=5;
Xset=ones(100,2);
E=ones(100);
while(delta > .0001)
iter=iter+1;
    [EmaxEmin
y]=maxbox2(x,d);
delta=Emax-Emin;
    x=y;
Xset(iter,:)=y;
E(iter)=Emax;
end
disp(Xset);
plot(Xset);
fori=1:iter
subplot(1,2,1);
holdon;
plot(Xset(i,1),Xset(i,2),
'*');
holdoff;
end
subplot(1,2,2);
plot(E(1:iter));
Xmax=y;
fmax=Emax;
iterTime=toc;
end
function [EmaxEmin  y]=
maxbox2(x,d)
Val=Eff2(x);
if Val < 1.0
x1=[x(1)+d x(2)];
    E1=Eff2(x1);
x2=[x(1)+2*d x(2)];
    E2=Eff2(x2);
x3=[x(1) x(2)+d];
    E3=Eff2(x3);
x4=[x(1) x(2)+2*d];
    E4=Eff2(x4);
    E=[E1;E2;E3;E4];
    [Emax,I]=max(E);
Emin=min(E);

y=eval(strcat('x',num2str
(I)));
end
end
```

The output is as follows:

$$\text{fmax} = 0.5345, \text{Xmax} = 2.74003.0000, \text{iter} = 37, \text{iterTime} = 0.2862$$

4.3 Hooke's Jeeves Pattern Search

In this optimization method, the combination of exploratory and heuristic move is used to find the optimum value. The initial starting point is chosen as (x,y) = (2,3). The delta and alpha values are chosen as 0.1 and 2. The Matlab code is given below (Fig. 3)

```
function
[funcmaxXmaxitertimespent
]=hj(X,delta,alpha)
tic;
clc;
cleardata;
delta1=delta;
f=ones(100);
fnext=0;
iter=0;
fmax=.1;
funcmax=0;
while (abs(funcmax-
fmax)>.0001)
iter=iter+1;
funcmax=fmax;
    [fmaxXmax
t]=expl(X,delta1);
if t==1
        X1=X;
X2=Xmax;[fnextXnext]=patt
ern(X1,X2,alpha);
else
delta1=delta/2;
end
X=Xmax;
holdon;
subplot(2,1,1);
plot(X(1),X(2),'*');
title('Hookes Jeeves Plot
of X(1), X(2)');
holdoff;
f(iter)=funcmax;
timespent=toc;
end
subplot(2,1,2);
plot(f(1:iter));
end
function
[fnextXnext]=pattern(X1,X
2,alpha)
test=1;
count=0;
while (test==1)
count=count+1;
Xnext=X2+alpha*(X2-X1);
if ((fx1(X2)>fx1(Xnext))
|| count==2)
Xnext=X2;
test=0;
else test=1;
end
X2=Xnext;
end
fnext=fx1(Xnext);
end
function [fmaxXmax t] =
expl(X, StepSize)
D=StepSize;
X1=[X(1)+D X(2)];
X2=[X(1) X(2)+D];
X3=[X(1)-D X(2)];
X4=[X(1) X(2)-D];
if (X3(1)<0)
    X3(1)=0;
end
if (X4(1) < 0)
    X4=0;
end
f=fx1(X);
f1=fx1(X1);
f2=fx1(X2);
f3=fx1(X3);
f4=fx1(X4);
[fmax I]=max([f f1 f2 f3
f4]);
if I==1
    t=0;Xmax=X;
else
t=1;Xmax=eval(strcat('X',
num2str(I-1)));
end
end
```

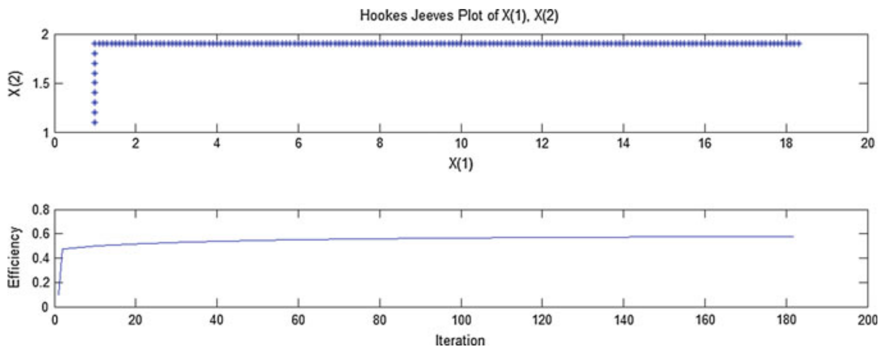

Fig. 3 The plot of (x,y) and f(x,y)

The output is as follows:

$$\text{funcmax} = 0.5772, \text{Xmax} = 17.70003.0000, \text{iter} = 157, \text{timespent} = 1.4124.$$

4.4 *Gradient Ascent Method (Cauchy's Method)*

Here, the search direction is the direction of gradient of the function at the point of evaluation in contrast to the negative of gradient in Gradient Descent Method. The Matlab Code is given below

```
function
[fgradmaxxoptitergradtime
]=gradDescent(xinit)
clc;
tic;
iter=0;
[dfx]=gradf(xinit);
f=ones(100);
xiter=ones(100,2);
[alpha]=gS(xinit,dfx);
x1=xinit-alpha*dfx;
Nfx=fx1(x1);
Ofx=fx1(xinit);
if (Nfx>Ofx)
xinit=x1;
end
while((Nfx-Ofx) > .00001)
iter=iter+1;
    [dfx]=gradf(xinit);
    [al-
pha]=gS(xinit,dfx);
x1=xinit-alpha*dfx;
Nfx=fx1(x1);
Ofx=fx1(xinit);
if (Nfx>Ofx)
xinit=x1;
end
f(iter)=Nfx;
xiter(iter,:)=x1;
disp(Nfx);
disp(Ofx);
end
xopt=x1;
fgradmax=Nfx;
gradtime=toc;
subplot(1,2,1);
plot(f(1:iter));
subplot(1,2,2);
plot(xiter((1:iter/100),1
),xiter((1:iter/100),2),'
*');
end
function [dfx]=gradf(x)
d=.001;
x1=[x(1)+d x(2)];
x2=[x(1)-d x(2)];
x3=[x(1) x(2)+d];
x4=[x(1) x(2)-d];
dfx(1)=(fx1(x2)-
fx1(x1))/(2*d);
dfx(2)=(fx1(x3)-
fx1(x4))/(2+d);
dfx=[dfx(1) dfx(2)];
end
function y=fx1(X)
y=(3*X(1)+4*X(2)+1)/(5*X(
1)+7*X(2)+5);
end
function [al-
pha]=gS(x,dfx)
a1=1;
a0=0;
ah=a1/2;
x1=x+a1*dfx;
xh=x+ah*dfx;
x0=x+a0*dfx;
while ((x1-x0)>.001)
if(fx1(x1)<fx1(xh))
        a1=ah;
end
if(fx1(x0)<fx1(xh))
        a0=ah;
end
ah=a1/2;
x1=x+a1*dfx;
xh=x+ah*dfx;
x0=x+a0*dfx;
end
alpha=a1;
end
```

The plots of number of iterations and the function value with respect to the iterations are given below (Fig. 4).

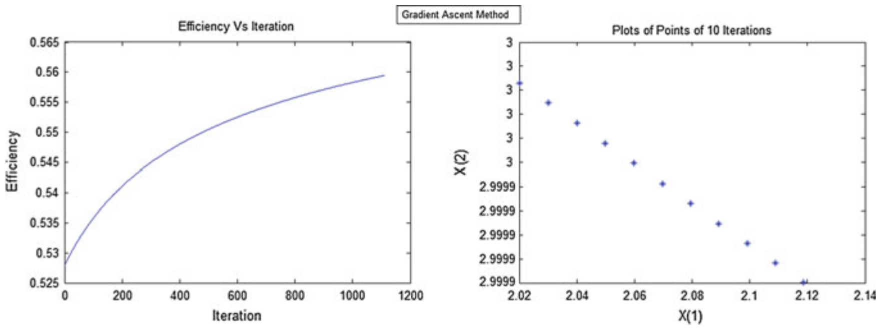

Fig. 4 The plots of (x,y) and f(x,y) against iterations

The output is as follows:

$$\text{gradmax} = 0.5595, \text{Xopt} = [7.63152.9965], \text{iterations} = 1113, \text{gradtime} = 0.3190$$

4.5 Genetic Algorithm

In case of genetic algorithm, a sample population in feasible region is selected at random. The crossover and mutation within the sample population is performed to get a new population based on the fitness level of sample population. The fitness function is the criteria for survival in subsequent generation. Here, the value of the objective function or Q(x) is the criteria for fitness. The sample population size depends on the accuracy of result expected, i.e., number of bits required for encoding the variable. We assume two decimal place accuracy of the variable. With the increase in number of variables, the cross over and mutation operations become complicated. The termination criterion has been chosen as difference of two successive iterations less than 10^{-4}. The population size has been chosen as 10. The Matlab code is given below (Fig. 5)

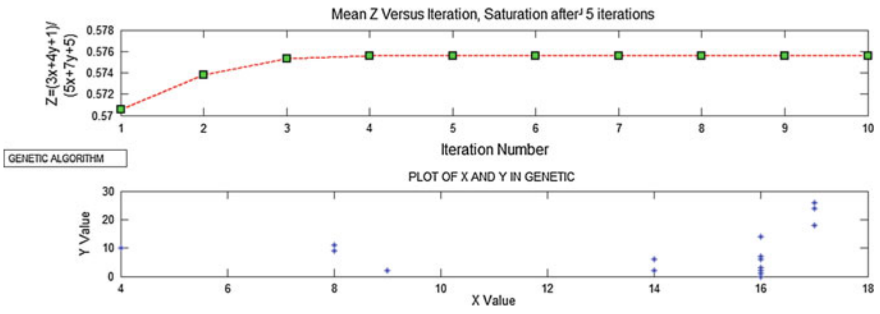

Fig. 5 Plot of output of genetic algorithm

```
function
[ZmaxiterXmaxYmaxtimespen
t]=callgeneFrac(x,y,tol,m
ax_iter)
clc;
tic;
r=randi([8,10]);
k=5;
output=ones(2,10,200);
[x1,y1,z,e,scat]=geneFrac3(x,y,r
,5);
out=ones(1,10);
mut=ones(1,10);
zmean=mean(z);
emean=mean(e);
p=0;
while(abs(zmean-emean)>tol)
    r=randi([9,10]);
    p=p+1;
if (p>max_iter/2)
        k=k+1;

[x1,y1,z,e,scat]=geneFrac3(x,y,r
,k);
end
if(p>2*max_iter/3)
        k=k+1;
[x1,y1,z,e,scat]=geneFrac3(x,y,r
,k);
end
if (p>max_iter)
break;
end
    x=x1;
    y=y1;

[x1,y1,z,e,scat]=geneFrac3(x,y,r
,5);
output(:,:,p)=[x1;y1];
zmean=mean(z);emean=mean(e);
out(p)=zmean;
    [Zmax m]=max(z);
Xmax=x1(m);
Ymax=y1(m);
end
for k=p+1:10
out(k)=zmean;
end
iter=p;
timespent=toc;
figure;
subplot(2,1,1);
plot(out,'--
rs','LineWidth',2,'MarkerEdgeCol
or','k','MarkerFaceColor','g','M
arkerSize',10);
xlabel('Iteration
Number','FontSize',12);
ylabel('Z=(3x+4y+1)/(5x+7y+5)','
FontSize',12);
title(strcat('Mean Z Versus It-
eration, Saturation
after',blanks(4),
iter),'FontSize',12);
for k=1:p
for l=1:10
holdon;
subplot(2,1,2);
plot(output(1,l,k),output(2,l,k)
,'*');
title('PLOT OF X AND Y IN
GENETIC');
holdoff;
end
end
end
```

The output is as follows:

$$\text{Zmax} = 0.5765, \text{iter} = 4, \text{Xmax} = 16, \text{Ymax} = 0, \text{timespent} = 0.5549$$

5 Result

This paper is intended to compare five algorithms. The five algorithms were run with the following Matlab code:

```
tol=.001;max_iter=100;
clc;
disp('Genetic
Algorithm');
[ZmaxiterXmaxYmaxtimespen
t]=callgeneFrac(x,y,tol,m
ax_iter);
tgene=timespent;
itergene=iter;
disp('Hookes Jeeves');
[funcmaxXmaxitertimespent
]=hj([1 1],.1,2);
thj=timespent;
iterhj=iter;
disp('Random Search');
[foptxopttimespent]=rando
mOptim(1000);
trs=timespent;
maxpts=1000;
disp(' Box Evaluation Method');
[fmaxXmaxiteriterTime]=bo
xEval([1 1],0.01);
tbox=iterTime;
iterbox=iter;
disp('Gradient Ascent
Method');
[fgradmaxgraditergradtime
]=gradDescent([1 1]);
fprintf('Computation Time
Genetic=%f, Hookes=%f,
Random Search=%f,
BoxEval=%f Grad=%f\n',
tgene,thj,trs,tbox,gradti
me);
fprintf(' Optimum values Genetic=%f,
Hookes=%f, Random Search=%f, BoxEval=%f
Grad=%f\n',
Zmax,funcmax,fopt,fmax,fg
radmax);
fprintf('Iterations Ge-
netic=%d, Hookes=%d, Ran-
dom
Search=%d,BoxEval=%d,Grad
=%d\n',itergene,iterhj,ma
xpts,iterbox,graditer);
```

The comparative result of running five algorithms individually on the same fractional function with almost same termination criteria and same initial starting point is given below in tabular form (Table 1 and Fig. 6).

Table 1 Comparative figures for various algorithms

Algorithms	Computation time	Iterations	Optimum values	Computation time/iteration
Random search	0.013811	1000	0.47016	0.138113
Box's evolution	0.154882	231	0.536978	6.704833
Hooke's Jeeves	0.14433	191	0.5779115	7.7556552
Gradient ascent	0.186542	1213	0.562720	1.537856
Genetic	1.4901413	52	0.55645161	286.565718

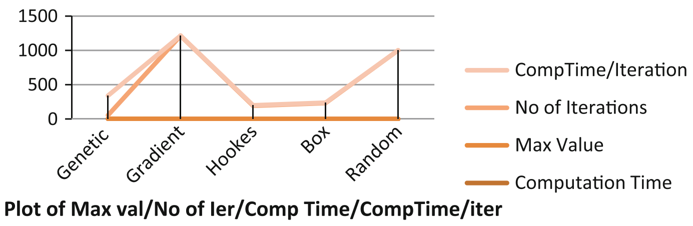

Fig. 6 Plot of (computation time/iteration)/optimum value/time per iteration for different types of algorithms

6 Conclusion

From the table of comparative performances, it is clear that computation time for genetic algorithm is highest, while random search takes lowest time. Strangely, Hooke's Jeeves Pattern Search takes lesser time to execute than Box's Evolution and others except Random Search. There is no doubt that the performance of genetic algorithm is the best. Though Gradient Descent is known to be a better algorithm than non gradient algorithms, Hooke's Jeeves algorithm performs better than Gradient Search in terms of computation time. In regard to number of iterations to reach termination, again genetic algorithm performs better than all other algorithms. The worst is the Random Search Algorithm. The performance of Hooke's Jeeves Method is better than Box's Evolutionary algorithm. This compelled the derivation of computation time per iteration. From the computation time per iteration, it is found that Gradient Descent Algorithm is having the best result, whereas genetic algorithm is the worst. This may be due to the fact that more complex is the algorithm, the computation time per iteration is higher.

The most interesting feature in Hooke's Jeeves algorithm is that it is giving the highest value in comparison to other algorithms. Gradient Descent Algorithm is giving the next highest. The performance of Random Search in terms optimum value is worst and highest in case of Hooke's Jeeves Algorithm.

The fractional programming is NP-hard problem. As a result, computation of algorithmic complexity is difficult. This paper does not intend to find the complexity. However, in order to make comparative assessment, Algorithmic Index is defined as follows:

$$\eta = \Delta_1(\mathrm{CT}) + \Delta_2(\mathrm{It}) + \Delta_3(\mathrm{Opt}) + \Delta_4(\mathrm{CT/It}). \tag{7}$$

CT	Computation Time
It	Iterations
Opt	Optimum Values
CT/It	Computation Time per Iteration
$\Delta = (\Delta_1 \Delta_2 \Delta_3 \Delta_4)$	Relative gewicht vector

Computation of Gewicht Vector: In order to estimate the gewicht vector, the following algorithm has been formulated:

```
Step 1   Choose random set of initial starting points.
Step 2   Find optimum parameters.
Step 3   Set a heuristics for derivation of Algorithmic Index.
Step 4   Performance Matrix is computed as follows:
         Performance Matrix = [$CT_{Algo}$ $It_{Algo}$ $Opt_{Algo}$ $(CT/It)_{Algo}$],
         , where Algo = Gentic Algorithm, Gradient Ascent Algorithm,
         Hooke's Jeeves, Box, and Random.
 Step5   Normalize Performance Matrix.
 Step6   Derive Algorithmic Index as follows:
         AlgoIndex = GewichtVector * Normalized Performance Matrix
 Step7   Find the set of gewicht for which the heuristics satisfies.
 Step8   The average gewicht over the set is computed.
 Step9   If the average gewicht does not satisfy heuristics, readjust heuristics
         by going to step2, and recompute till gewicht satisfies heuristics.
Step10   Estimated gewicht vector is derived.
         Step11—Apply estimated gewicht vector to the Normalized
         Performance Matrix on a new random initial starting point. Check
         whether heuristics is satisfied.
```

The MATLAB code for estimation of gewicht vector is given in supplementary material. The estimated gewicht turns out to be $(\varDelta_1\varDelta_2\varDelta_3\varDelta_4) = (0.237586\,0.292759\,0.358276\,0.111379)$

With the random starting point figures, the performance matrix turns out to be Table 2

After normalizing the performance index and using estimated gewicht vector, algorithmic index turns to be:

Table 3

In a nutshell, it may be concluded that genetic algorithm performs far better than other algorithms. The algorithmic index is highest for genetic algorithm. The ratio of Ƞ

Table 2 Performance matrix

Performance matrix					
	Genetic	Gradient	Hookes	Box	Random
Computation time	1.49014173	0.186542	0.14433	0.154882	0.013811
Max value	0.55645161	0.56272	0.577915	0.536978	0.47016
No of iterations	52	1213	191	231	1000
CompTime/iteration	286.565718	1.537856	7.556552	6.704833	0.138113

Table 3 Normalised performance matrix

Normalized performance matrix					
	Genetic	Gradient	Hookes	Box	Random
Computation time	107.892569	13.50643	10.45011	11.21409	1
Max value	0.96286003	0.973707	1	0.929164	0.813544
No of iterations	1	23.32692	3.673077	4.442308	19.23077
CompTime/iteration	2074.85709	11.13473	54.71263	48.54583	1
Algorithmic index	257.37	13.092	10.185	9.935	7.477

values of genetic and gradient, the nearest competitor is approximately 19.658. That means, performance of genetic algorithm is nearly 20 times better than the remaining. The Gradient Ascent performs better than Hooke's Jeeves and Box's Algorithm. Here, two variable fractional functions have been studied. The research may be extended to cases of higher dimensions.

Acknowledgements Special thanks go to my guides Dr. Sujyasikha Das and Dr. Swarup Prasad Ghosh for inspiring me to write the paper and implementing the scenario in Matlab. The paper would remain unfinished if I don't convey my regards and heartfelt thanks to Dr. Nabendu Chaki for relentless support to my academics. He has been the driving force for all the activities.

Appendix

Algorithm for Random Search

```
Step 1: Choose initial $x^0$, $z^0$, $\epsilon$ such that the minimum lies in ($x^0 - 1/2z^0$, $x^0 + 1/2z^0$). For each Q block, set q = 1 and p = 1.
Step 2: For i = 1,2…N, create points using uniform distribution of m in the range (–0.5,0.5). Set $x_i^{(p)} = x_i^{q-1} + mz_i^{q-1}$.
Step 3: If $x^{(p)}$ is infeasible and p < P, repeat Step 2. If $x^{(p)}$ is feasible, save $x^{(p)}$ and $f(x^{(p)})$. Increment p and repeat step 2;
        Else if p = P, set $x^q$ to be the point that has lowest $f(x^{(p)})$ over all feasible $x^{(p)}$ including $x^{q-1}$
        And reset p = 1.
Step 4: Reduce the range via $z_i^q = \epsilon\, z_i^{q-1}$.
Step 5: If q > Q. Stop.
        Else increment q and continue to Step 2.
```

Box's Evolutionary Algorithm

```
Step 1: Choose initial point. Choose size reduction step $\delta_i$ and termination criteria $\epsilon$.
Step 2: If $\delta_i < \epsilon$ STOP.
Step 3: Else create $2^N$ points by adding and subtracting $\delta_i$ from each variable at the initial point.
```

Step 4: Compute function values at all 2^N points. Find the optimum among these points. Set it as initial point for next iteration.
Step 5: Reduce size of the step to $\delta_i/2$ and go to Step 2.

Hooke's Jeeves' Algorithm

Step 1: Initial point is selected and objective function is evaluated.
Step 2: Search is made in the direction of each dimension by a step size S_i to find lowest of functional value.
Step 3: In case the function value does not decrease in any direction, the step size is reduced and fresh search is made.
Step 4: If the value of objective function reduces, a new initial point is found as follows:
$X_{i,o}^{(k+1)} = X_i^{k+1} + \theta(X_i^{k+1} - X_i^k)$, $\theta > 1$.
Step 5: This search continues till the termination criteria is met, i.e., $\theta < \epsilon$.

Gradient Descent Method

Step 1: Choose initial point $x^{(0)}$ and termination parameters ϵ_1 and ϵ_2.
Step 2: Compute first derivative $\nabla f(x^k)$.
Step 3: If $||\nabla f(x^k)|| \leq \epsilon_1$ STOP.
Else go to next step
Step 4: By unidirectional search, find α^k such that $f(x^{(k+1)}) = f(x^k - \alpha^k \nabla f(x^k))$ is minimum. One criteria for termination is $| \nabla f(x^{k+1}).\nabla f(x^k) | \leq \epsilon_2$.
Step 5: If $\frac{||x^{k+1} - x^k||}{||x^k||} \leq \epsilon_1$, then STOP.
Else set $k = k + 1$, go to step 2.

Genetic Algorithm

Start and generate a random population of size n.
Fitness: Evaluate fitness of each chromosome.
New Population: Create new population by repeating the steps below
Select two parent chromosomes from the population according to best fitness.
Cross over the parents, with a crossover probability to form new population.
With a mutation probability, mutate the new offspring.
Add the new offspring in the population.
Replace: Use the new generation for next iteration.
Test: Check termination criteria.
Loop: Go to step 2.

MATLAB Code for Estimating Gewicht Vector

```
ExtFiveMethods.m
function []=ExtFiveMethods()
  % for computing Parameters
 x=rand(1,10);
 y=rand(1,10);
    [avS avAlgoIndex]=pmComp(x,y);
    x=rand(1,10);
    y=rand(1,10);
    fn='fiveMethod.mat';
    [NPM1 PM]=FiveMethods(x,y);
    save(fn,'NPM1','PM');
    AlgoIndex=avS*NPM1;
    fn1='fiveMethod1.mat';
    save(fn1,'avS','AlgoIndex');
    disp('Estimated Algorithimic Index');
    fprintf('%1.3f %1.3f %1.3f %1.3f %1.3f\n',AlgoIndex);
    fprintf('\n');
end
pmComp.m
function [avS avAlgoIndex]=pmComp(x,y)
x1=x;
y1=y;
[NPM perfMatrix]=FiveMethods(x1,y1);
paramSt=ones(50,4);
t=1;
   for k=0.01:.1:1
        for j=0.01:.1:1
             for i=0.01:.1:1
                 l=1-i-j-k;
                 param=[i j k l];
                 if ((l<=0)||(k==1)||(j==1)||(i==1)||(i==1))
                     break;
                 end
                  AlgoIndex=param*NPM;
                  if ((i<1) || (j<1) || (k<1) || (l<1))
                        if( (AlgoIndex(1,1)>AlgoIndex(1,2)) &&
...(AlgoIndex(1,2)>AlgoIndex(1,3)) &&
(AlgoIndex(1,3)>AlgoIndex(1,4))...
                   && (AlgoIndex(1,4)>AlgoIndex(1,5)) )
                    paramSt(t,:)=param;
                            t=t+1;
                      end
                 end
```

```
            end
        end
    end
    comps=sum(paramSt(1:t-1,:));
    avS=(1/(t-1))*comps;
    avAlgoIndex=avS*NPM;
    pause;
    end
FiveMethods.m
function [NPM perfMatrix]=FiveMethods(x,y)
tol=.001;max_iter=100;
[Zmax iter Xmax Ymax timespent]=callgeneFrac(x,y,tol,max_iter);
tgene=timespent;
itergene=iter;
 [funcmax Xmax iter timespent]=hj([x(1) y(1)],.1,2);
thj=timespent;
iterhj=iter;
[fopt xopt timespent]=randomOptim(1000);
trs=timespent;
maxpts=1000;
[fmax Xmax iter iterTime]=boxEval([x(1) y(1)],0.01);
tbox=iterTime;
iterbox=iter;
[fgradmax xopt graditer gradtime]=gradDescent([1 1]);
format;
perfMatrix=ones(4,5);
perfMatrix(1,:)=[tgene gradtime thj tbox  trs ];
perfMatrix(2,:)=[Zmax fgradmax funcmax fmax  fopt  ];
perfMatrix(3,:)=[itergene graditer iterhj iterbox  maxpts ];
perfMatrix(4,:)=10000*[tgene/itergene  gradtime/graditer
thj/iterhj tbox/iterbox trs/maxpts ];
tmin=min(perfMatrix(1,:));
OutMax=max(perfMatrix(2,:));
iterMin=min(perfMatrix(3,:));
perIterMin=min(perfMatrix(4,:));
NPM=ones(4,5);
NPM(1,:)=(1/tmin)*perfMatrix(1,:);
NPM(2,:)=(1/OutMax)*perfMatrix(2,:);
NPM(3,:)=(1/iterMin)*perfMatrix(3,:);
NPM(4,:)=(1/perIterMin)*perfMatrix(4,:);
End
```

References

1. Charnes, A.C.W.: An explicit general solution in linear fractional programming. Naval Res. Logist. Quart. **20**, 449–467 (1973)
2. Bitran, G.R., Novaes, A.G.: Linear programming with fractional objective function. Oper. Res. **21**, 22–29 (1973)
3. Birtan, G.R., Magnanti, T.L.: Duality and sensitivity analysis of fractional objective function. Oper. Res. **24**, 675–699 (1976)
4. Swarup, K.: Linear fractional programming. Oper. Res. **13**(6), 1029–1036 (1965)
5. Dantzig, G.B.: Linear Programming under uncertainty. Manage. Sci. **1**(3 and 4), 197–206 (1955)
6. Dantzig, G.B., Mandansky, A.: on solution of two stage linear programming under uncertainty. Barkley Symp. Maths Stat. **1**(3 and 4), 165–176 (1961)
7. Bazalinov, E.B.: Linear Fractional Programming. Kluwer Academic Publishers, Dordrecht (2003)
8. Bajalinov, E.: Scaling problems in linear fractional programming. In: proceeding of 10th international conference on operation research, vol. 3, no. 1, pp. 22–24 (2004)
9. Shohrab, E., Morteza, P.: Solving the Interval Valued Fractional Programming. Am. J. Comput. Math. **2**(1), 51–55 (2012)
10. Barros, A.I., Frenk, J.B., Schaible, S., Zhang, S.: A new algorithm for generalised fractional programming. Math. Program. **72**(2), 147–175 (1996)
11. Gogia, N.: Revised simplex algorithm for linear fractional programming problem. Math. Student **36**(1), 55–57 (1969)
12. Horvath, I.: AsupraprogramliriifracJionare lineare cu restricJii suplimentare. Informatica pentru Conducere, pp. 101–102 (1981)
13. Stahl, J.: Two new methods for solution of hyperbolic programming. In: Publications of the Mathematical Institute of Hungarian Science, vol. 9, no. B, pp. 743–754 (1964)
14. Stancu-Minasian, I.M.: Stochastic Programming with MultiObjective Function. D. Reidel Publishing Company, Dordrecht (1984)
15. Buhler, W.: A note on fractional interval programming. Oper. Res. A-B **19**, 1, 29–36 (1975); **Z**(19), 29–35 (1975)
16. Robins, H., Monro, S.: A stochastic approximation method. Ann. Math. Stat. **22**, 400–407 (1951)
17. Costa, A., Jones, O., Kroese, D.: Convergence properties of the cross entropy method for discrete optimization. Oper. Res. Lett. **35**, 573–580 (2007)
18. Zhang, Q., Muhlenbein, H.: On the convergence of a class of estimation of distribution algorithm. IEEE Trans. Evol. Comput. **8**, 127–134 (2004)
19. Binglsley, P.: Convergence of probability measures. John Wiley and Sons, New York (1999)
20. Box, G.E.: Evolutionary operation: a method for increasing industrial productivity. Appl Stat **6**, 81–101 (1957)
21. Brent, R.P.: Algorithms for Minimization without derivatives. Printice Hall, EngleWoods Cliffs (2002)
22. Mifflin, R., Strodiot, J.J.: A Bracketing technique to ensure desirable convergence in univariate minimisation. Math. Prog. **17**, 100–117 (1975)
23. Mifflin, R., Strodiot, J.J.: A rapidly convergent five-point algorithm for univariate minimisation. Math. Prog. **62**, 299–319 (1993)
24. Box, G.P., Wilson, K.B.: On the experimental attainment of optimal conditions. Stat. Soc. **13**, 1–13 (1951)
25. Box, G.E.P., Draper, N.R.: Evolutionary Operation: A Statistical Method For Process Improvement. Wiley, New York (1998)
26. Hooke, R., Jeeves, T.A.: Direct search solution for numerical and statistical problem. ACM **8**, 212–219 (1961)

27. Nelder, J.A., Mead, R.: A simplex method for function minimisation. Comput. J. **7**, 308–313 (1965)
28. Fletcher, R.: Practical Methods for Optimisation. John Wiley and Sons, Chichester (1987)
29. Murray, W., Wright, M.H., Gill, P.E.: Practical Optimization. Academic Press, London (1981)
30. Gabay, D.: Reduced Quasi Newton method with feasibilty improvement for nonlinear constrained optimisation. Math. Prog. Stud. **16**, 18–44 (1982)
31. Fletcher, R.: Conjugate Gradient Methods for Indefinite Systems. Numer. Anal. Rep. **5**, 11 (1975)
32. Zangwill, W.: Nonlinear Programming: A Unified Approach. Printice Hall, Englewood Cliffs (1969)
33. Holland, J.H.: Hierarchical description of universal spaces and adaptive systems. Tech. Rep ORA Project 01252 (1968)
34. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
35. Grefenstette, J.J.: Deception considered harmful. In: Whitley, L.D. (ed.) Foundations of genetic algorithms (1993)
36. Fogel, D.G.: Schema processing under proportional selection in the presence of random effects. IEEE Trans. Evol. Comput. **1**(4), 290–293 (1997)
37. Radcliffe, N.J.: Schema Processing. In: Handbook of evolutionary computation, pp. B2.5–1.10. Oxford University Press (1997)