Липецкий государственный технический университет

Кафедра прикладной математики

КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ МАТЕМАТИЧЕСКИХ ИССЛЕДОВАНИЙ

Лекция 2

Оптимизация в R

Составитель - Сысоев А.С., к.т.н., доцент

Липецк - 2021

Outline

1.3. Популяционные процедуры поисковой оптимизации

1.4. Генетические и эволюционные алгоритмы

1.5. Дифференциальная эволюция

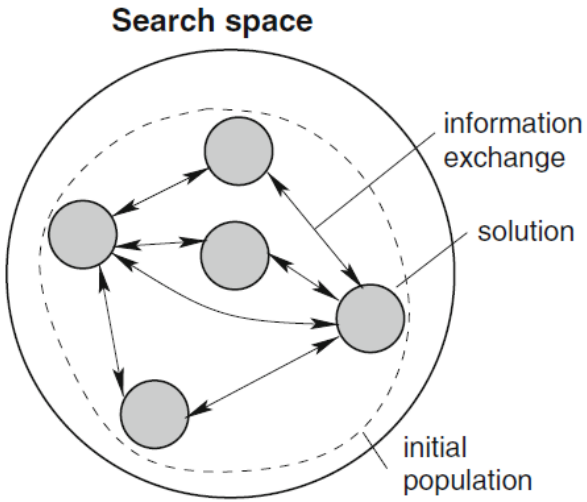
1.6. Роиные алгоритмы

1.7. Estimation of Distribution Algorithm

1.8. Сравнение популяционных алгоритмов

1.9. Многоцелевая оптимизация

1.3. Популяционные процедуры поисковой оптимизации



- Локальный поиск – в окрестности одной точки
- Популяционный поиск – генетические алгоритмы, эволюционные алгоритмы, …
- Больше вычислительных усилий, но: быстрее сходимость.
- Алгоритмы различаются: как представлены решения и какие атрибуты с ними связаны, как строится новое решение.

- В основном механизмы построения решений заимствованы из природы: генетика, естественный отбор, коллективное поведение животных и растений.

1.4. Генетические и эволюционные алгоритмы

Эволюционные вычисления (ЭВ) лежат в основе нескольких алгоритмов оптимизации и основаны на феномене естественного отбора и принципа состязательности.

Генетические алгоритмы (ГА) в самом начале оперировали только бинарным представлением решения с применением процедуры кроссовера для получения нового решения. Позднее идеи ЭВ были применены для построения ГА с целью работы с исходным представлением решения задачи и регулирования генетических операторов от кроссовера до простой мутации.

Биологическая терминология:

Особь, популяция

Генотип, геном, хромосома – структура особи, популяции

Ген – позиция в хромосоме, аллель – значение гена

Фенотип – совокупность признаков, оценка, насколько особь «хороша» (целевая функция)

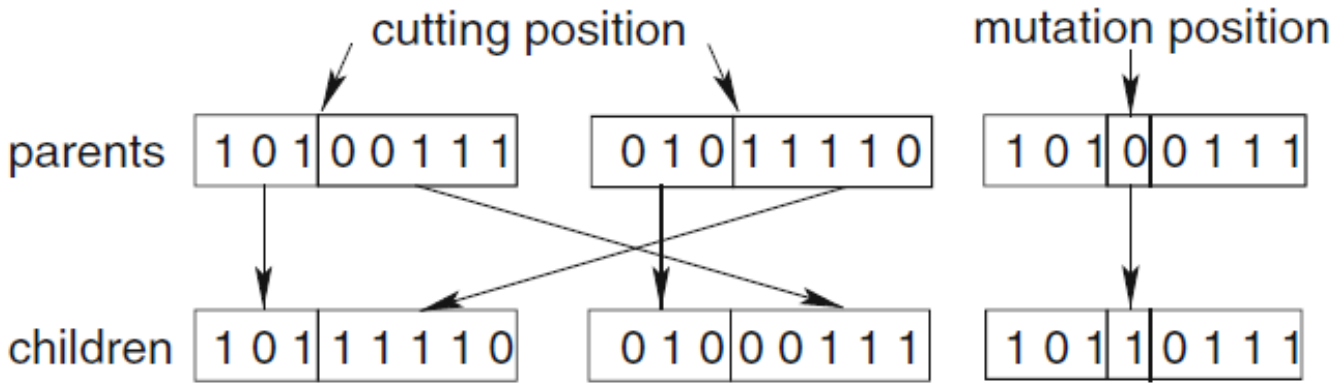
Размножение – совокупность кроссовера и мутаций. Участвуют два предка, получается потомок, отличный от остальных из-за мутации.

1.4. Генетические и эволюционные алгоритмы

Genetic/evolutionary algorithm as implemented by the `genalg` package

1: **Inputs:** f, C ▷ f is the evaluation (fitness) function, C includes control parameters
2: $P \leftarrow initialization(C)$ ▷ random initial population
3: $N_P \leftarrow get\_population\_size(C)$ ▷ population size
4: $E \leftarrow get\_elitism(C)$ ▷ number of best individuals kept (elitism)
5: $i \leftarrow 0$ ▷ i is the number of iterations of the method
6: **while** $i < maxit$ **do**
7: $F_P \leftarrow f(P)$ ▷ evaluate current population
8: $P_E \leftarrow best(P, F_P, E)$ ▷ set the elitism population (lowest E fitness values)
9: $Parents \leftarrow selectparents(P, F_P, N_P - E)$ ▷ select $N_P - E$ parents from current population
10: $Children \leftarrow crossover(Parents, C)$ ▷ create $N_P - E$ children solutions
11: $Children \leftarrow mutation(Children, maxit, i)$ ▷ apply the mutation operator to the children
12: $P \leftarrow E \cup Children$ ▷ set the next population
13: $i \leftarrow i + 1$
14: **end while**
15: **Output:** P ▷ last population

1.4. Генетические и эволюционные алгоритмы



Кроссовер **мутация**

Мутация «реальных» генов

$$g' = 0.67 \times r_d \times d_f \times R_g$$

$$r_d \in \{-1, 1\}$$

$$d_f = (maxit - i)/maxit$$

$$R_g = \max(g) - \min(g)$$

1.4. Генетические и эволюционные алгоритмы

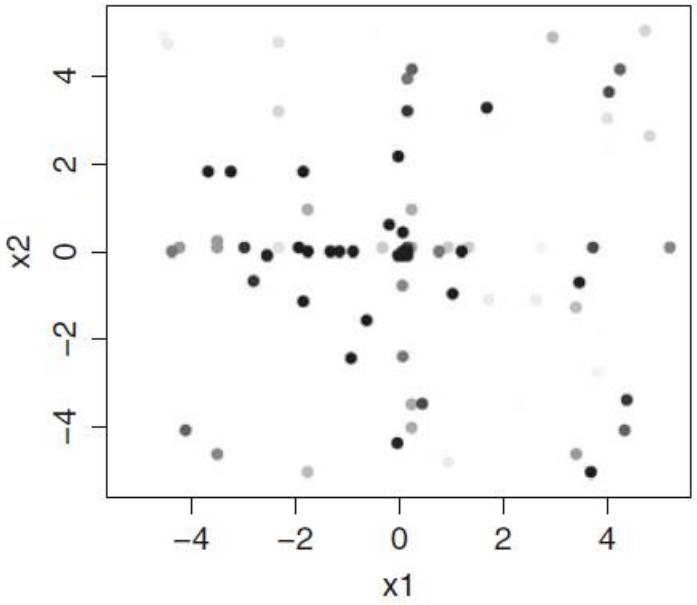
```
### bag-genalg.R file ###
library(genalg) # load genalg package
source("functions.R") # load the profit function

# genetic algorithm search for bag prices:
D=5 # dimension (number of prices)
MaxPrice=1000
Dim=ceiling(log(MaxPrice,2)) # size of each price (=10)
size=D*Dim # total number of bits (=50)
intbin=function(x) # convert binary to integer
{ sum(2^(which(rev(x==1))-1)) } # explained in Chapter 3
bintbin=function(x) # convert binary to D prices
{ # note: D and Dim need to be set outside this function
  s=vector(length=D)
  for(i in 1:D) # convert x into s:
  { ini=(i-1)*Dim+1;end=ini+Dim-1
    s[i]=intbin(x[ini:end])
  }
  return(s)
}
bprofit=function(x) # profit for binary x
{ s=bintbin(x)
  s=ifelse(s>MaxPrice,MaxPrice,s) # repair!
  f=-profit(s) # minimization task!
  return(f)
}
```

1.4. Генетические и эволюционные алгоритмы

```
# genetic algorithm execution:
G=rbga.bin(size=size,popSize=50,iters=100,zeroToOneRatio=1,
    evalFunc=bprofit,elitism=1)
# show results:
b=which.min(G$evaluations) # best individual
cat("best:",bintbin(G$population[b,]),"f:",-G$evaluations[b],
    "\n")
pdf("genalg1.pdf") # personalized plot of G results
plot(-G$best,type="l",lwd=2,ylab="profit",xlab="generations")
lines(-G$mean,lty=2,lwd=2)
legend("bottomright",c("best","mean"),lty=1:2,lwd=2)
dev.off()
summary(G,echo=TRUE) # same as summary.rbga

> source("bag-genalg.R")
best: 427 431 425 355 447 f: 43671
GA Settings
  Type                  = binary chromosome
  Population size       = 50
  Number of Generations = 100
  Elitism               = 1
  Mutation Chance       = 0.0196078431372549

Search Domain
  Var 1 = [,]
  Var 0 = [,]

GA Results
  Best Solution : 0 1 1 0 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0 1 1
      0 1 0 1 0 0 1 0 1 0 1 1 0 0 0 1 1 0 1 1 0 1 1 1 1 1 1
```

1.4. Генетические и эволюционные алгоритмы

```
### sphere-genalg.R file ###
library(genalg) # load genalg

# evolutionary algorithm for sphere:
sphere=function(x) sum(x^2)
D=2
monitor=function(obj)
{ if(i==1)
    { plot(obj$population,xlim=c(-5.2,5.2),ylim=c(-5.2,5.2),
           xlab="x1",ylab="x2",type="p",pch=16,
           col=gray(1-i/maxit))
    }
  else if(i%%K==0) points(obj$population,pch=16,
                          col=gray(1-i/maxit))
  i<<-i+1 # global update
}

maxit=100
K=5 # store population values every K generations
i=1 # initial generation
# evolutionary algorithm execution:
pdf("genalg2.pdf",width=5,height=5)
set.seed(12345) # set for replicability purposes
E=rbga(rep(-5.2,D),rep(5.2,D),popSize=5,iters=maxit,
       monitorFunc=monitor,evalFunc=sphere)
b=which.min(E$evaluations) # best individual
cat("best:",E$population[b,],"f:",E$evaluations[b],"\n")
dev.off()
```




```
> source("sphere-genalg.R")
best: 0.05639766 0.009093091 f: 0.00326338
```

1.5. Дифференциальная эволюция

- Используются арифметические операции при построении нового решения
- Выбор трех особей (s_1, s_2, s_3)
- Создание мутанта

$$s_{m,j} = s_{1,j} + F \times (x_{2,j} - x_{3,j}), \; F \in [0, 2]$$

- Если созданный мутант выходит за границы, то

$$s_{m,j} = \max(s_j) - \mathcal{U}(0, 1)(\max(s_j) - min(s_j)) \qquad s_{m,j} > \max(s_j),$$

$$s_{m,j} = \min(s_j) + \mathcal{U}(0, 1)(\max(s_j) - \min(s_j)) \qquad s_{m,j} < \max(s_j),$$

- Новые потомки создаются до тех пор, пока все гены не мутируют или r > CR, r = U(0,1), CR – вероятность кроссовера.
- Новая популяция – полученные потомки и оставшиеся родители.

1.5. Дифференциальная эволюция

Differential evolution algorithm as implemented by the `DEoptim` package

Inputs: f, C ▷ f is the evaluation (fitness) function, C includes control parameters
$P \leftarrow initialization(C)$ ▷ set initial population
$B \leftarrow best(P, f)$ ▷ best solution of the initial population
$i \leftarrow 0$ ▷ i is the number of iterations of the method
while not $termination\_criteria(P, f, C, i)$ **do** ▷ DEoptim uses up to three termination criteria
  for each individual $s \in P$ **do** ▷ cycle all population individuals
    $s' \leftarrow mutation(P, C)$ ▷ differential mutation, uses parameters F and CR
    if $f(s') < f(s)$ **then** $P \leftarrow replace(P, s, s')$ ▷ replace s by s' in the population
    end if
    if $f(s') < f(B)$ **then** $B \leftarrow s'$ ▷ minimization goal
    end if
  end for
  $i \leftarrow i + 1$
end while
Output: B, P ▷ best solution and last population

1.5. Дифференциальная эволюция

```
### sphere-DEoptim.R file ###
library(DEoptim) # load DEoptim

sphere=function(x) sum(x^2)
D=2
maxit=100
set.seed(12345) # set for replicability
C=DEoptim.control(strategy=1,NP=5,itermax=maxit,CR=0.9,F=0.8,
                  trace=25,storepopfrom=1,storepopfreq=1)
# perform the optimization:
D=suppressWarnings(DEoptim(sphere,rep(-5.2,D),rep(5.2,D),
                           control=C))
# show result:
summary(D)
pdf("DEoptim.pdf",onefile=FALSE,width=5,height=9,
    colormodel="gray")
plot(D,plot.type="storepop")
dev.off()
cat("best:",D$optim$bestmem,"f:",D$optim$bestval,"\n")
```

```
> source("sphere-DEoptim.R")
Iteration: 25 bestvalit: 0.644692 bestmemit:    0.799515
     0.073944
Iteration: 50 bestvalit: 0.308293 bestmemit:    0.550749
    -0.070493
Iteration: 75 bestvalit: 0.290737 bestmemit:    0.535771
    -0.060715
Iteration: 100 bestvalit: 0.256731 bestmemit:    0.504867
    -0.042906
***** summary of DEoptim object *****
best member   :  0.50487 -0.04291
best value    :  0.25673
after         :  100 generations
fn evaluated  :  202 times
************************************
best: 0.5048666 -0.0429055 f: 0.2567311
```

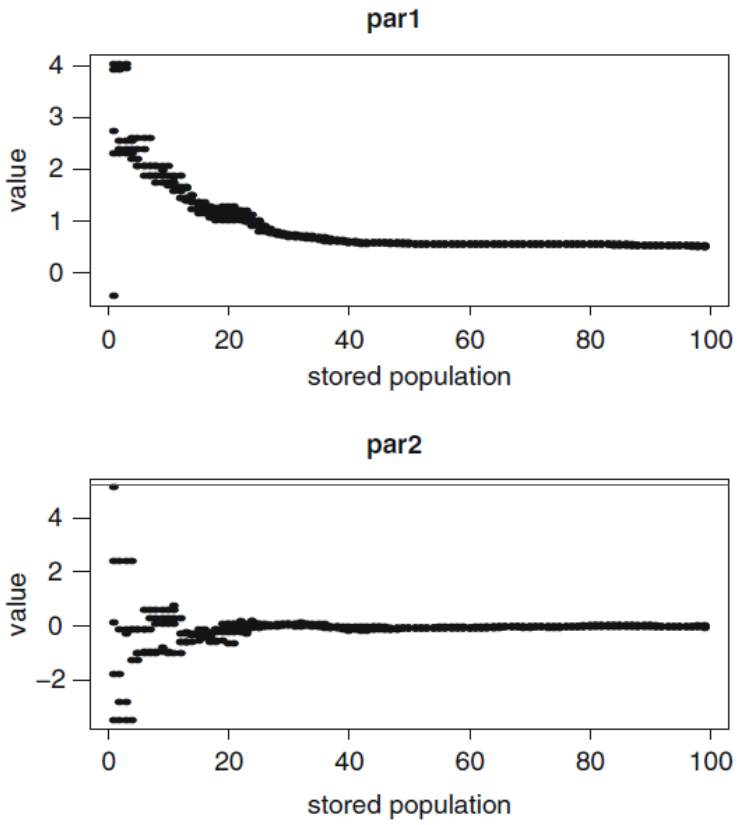
1.6. Роиные алгоритмы

Particle swarm optimization pseudo-code for SPSO 2007 and 2011

Inputs: f, C ▷ f is the fitness function, C includes control parameters
$P \leftarrow initialization(C)$ ▷ set initial swarm (topology, random position and velocity, previous best and previous best position found in the neighborhood)
$B \leftarrow best(P, f)$ ▷ best particle
$i \leftarrow 0$ ▷ i is the number of iterations of the method
while not $termination\_criteria(P, f, C, i)$ **do**
  for each particle $x = (s, v, p, l) \in P$ **do** ▷ cycle all particles
    $v \leftarrow velocity(s, v, p, l)$ ▷ compute new velocity for x
    $s \leftarrow s + v$ ▷ move the particle to new position s (*mutation*)
    $s \leftarrow confinement(s, C)$ ▷ adjust position s if it is outside bounds
    if $f(s) < f(p)$ **then** $p \leftarrow s$ ▷ update previous best
    end if
    $x \leftarrow (s, v, p, l)$ ▷ update particle
    if $f(s) < f(B)$ **then** $B \leftarrow s$ ▷ update best value
    end if
  end for
  $i \leftarrow i + 1$
end while
Output: B ▷ best solution

1.6. Роиные алгоритмы

```
### sphere-psoptim.R file ###
library(pso) # load pso

sphere=function(x) sum(x^2)

D=2; maxit=10; s=5
set.seed(12345) # set for replicability
C=list(trace=1,maxit=maxit,REPORT=1,trace.stats=1,s=s)
# perform the optimization:
PSO=psoptim(rep(NA,D),fn=sphere,lower=rep(-5.2,D),
            upper=rep(5.2,D),control=C)
# result:
pdf("psoptim1.pdf",width=5,height=5)
j=1 # j-th parameter
plot(xlim=c(1,maxit),rep(1,s),PSO$stats$x[[1]][j,],pch=19,
     xlab="iterations",ylab=paste("s_",j," value",sep=""))
for(i in 2:maxit) points(rep(i,s),PSO$stats$x[[i]][j,],pch=19
dev.off()
pdf("psoptim2.pdf",width=5,height=5)
plot(PSO$stats$error,type="l",lwd=2,xlab="iterations",
     ylab="best fitness")
dev.off()
cat("best:",PSO$par,"f:",PSO$value,"\n")
```



1.7. Estimation of Distribution Algorithm

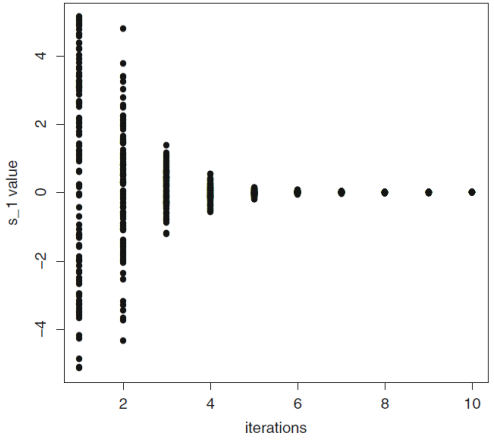
Generic EDA pseudo-code implemented in `copulaedas` package, adapted from Gonzalez-Fernandez and Soto (2012)

Inputs: f, C ▷ f is the fitness function, C includes control parameters (e.g., N_P)
$P \leftarrow initialization(C)$ ▷ set initial population (*seeding* method)
if $required$ **then** $P \leftarrow local\_optimization(P, f, C)$ ▷ apply local optimization to P
end if
$B \leftarrow best(P, f)$ ▷ best solution of the population
$i \leftarrow 0$ ▷ i is the number of iterations of the method
while not $termination\_criteria(P, f, C)$ **do**
    $P' \leftarrow selection(P, f, C)$ ▷ selected population P'
    $M \leftarrow learn(P')$ ▷ set probabilistic model M using a learning method
    $P' \leftarrow sample(M)$ ▷ set sampled population from M using a sampling method
    if $required$ **then** $P' \leftarrow local\_optimization(P', f, C)$ ▷ apply local optimization to P'
    end if
    $B \leftarrow best(B, P', f)$ ▷ update best solution (if needed)
    $P \leftarrow replacement(P, P', f, C)$ ▷ create new population using a replacement method
    $i \leftarrow i + 1$
end while
Output: B ▷ best solution

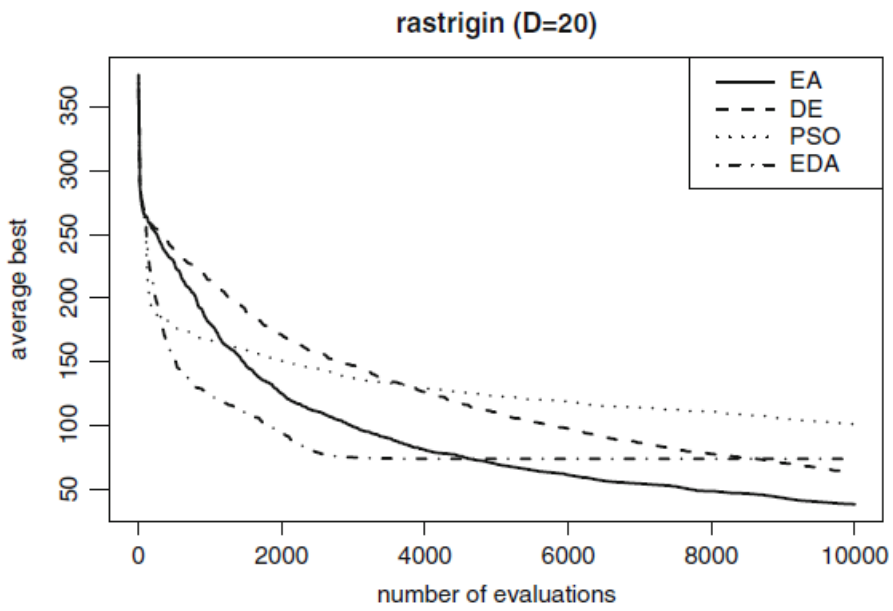
1.7. Estimation of Distribution Algorithm

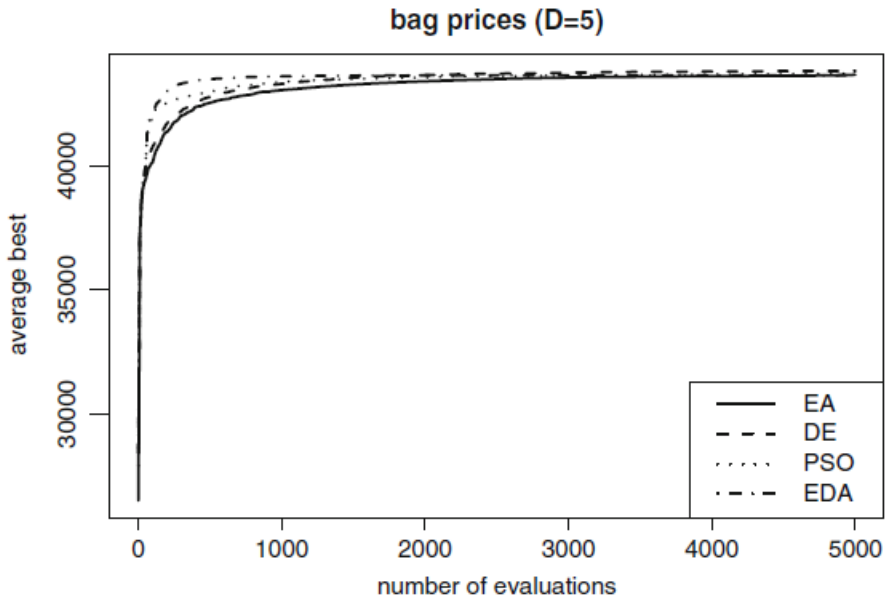
```
### sphere-EDA.R file ###
library(copulaedas)
sphere=function(x) sum(x^2)
D=2; maxit=10; LP=5
set.seed(12345) # set for replicability
# set termination criterion and report method:
setMethod("edaTerminate","EDA",edaTerminateMaxGen)
setMethod("edaReport","EDA",edaReportSimple)
# set EDA type:
UMDA=CEDA(copula="indep",margin="norm",popSize=LP,maxGen=maxit)
UMDA@name="UMDA (LP=5)"
# run the algorithm:
E=edaRun(UMDA,sphere,rep(-5.2,D),rep(5.2,D))
# show result:
show(E)
cat("best:",E@bestSol,"f:",E@bestEval,"\n")

# second EDA execution, using LP=100:
maxit=10; LP=100;
UMDA=CEDA(copula="indep",margin="norm",popSize=LP,maxGen=maxit)
UMDA@name="UMDA (LP=100)"
setMethod("edaReport","EDA",edaReportDumpPop) # pop_*.txt files
E=edaRun(UMDA,sphere,rep(-5.2,D),rep(5.2,D))
show(E)
cat("best:",E@bestSol,"f:",E@bestEval,"\n")

# read dumped files and create a plot:
pdf("eda1.pdf",width=7,height=7)
j=1; # j-th parameter
i=1;d=read.table(paste("pop_",i,".txt",sep=""))
plot(xlim=c(1,maxit),rep(1,LP),d[,j],pch=19,
     xlab="iterations",ylab=paste("s_",j," value",sep=""))
for(i in 2:maxit)
{ d=read.table(paste("pop_",i,".txt",sep=""))
  points(rep(i,LP),d[,j],pch=19)
}
dev.off()
```

```
> source("sphere-EDA.R")

  Generation      Minimum         Mean    Std. Dev.
           1 7.376173e+00 1.823098e+01 6.958909e+00
           2 7.583753e+00 1.230911e+01 4.032899e+00
           3 8.001074e+00 9.506158e+00 9.969029e-01
           4 7.118887e+00 8.358575e+00 9.419817e-01
           5 7.075184e+00 7.622604e+00 3.998974e-01
           6 7.140877e+00 7.321902e+00 1.257652e-01
           7 7.070203e+00 7.222189e+00 1.176669e-01
           8 7.018386e+00 7.089300e+00 4.450968e-02
           9 6.935975e+00 7.010147e+00 7.216829e-02
          10 6.927741e+00 6.946876e+00 1.160758e-02

Results for UMDA (LP=5)
Best function evaluation    6.927741e+00
No. of generations          10
No. of function evaluations 50
CPU time                    0.103 seconds
best: 1.804887 -1.915757 f: 6.927741

Results for UMDA (LP=100)
Best function evaluation    5.359326e-08
No. of generations          10
No. of function evaluations 1000
CPU time                    0.036 seconds
best: -0.00013545 0.0001877407 f: 5.359326e-08
```



1.8. Сравнение популяционных алгоритмов

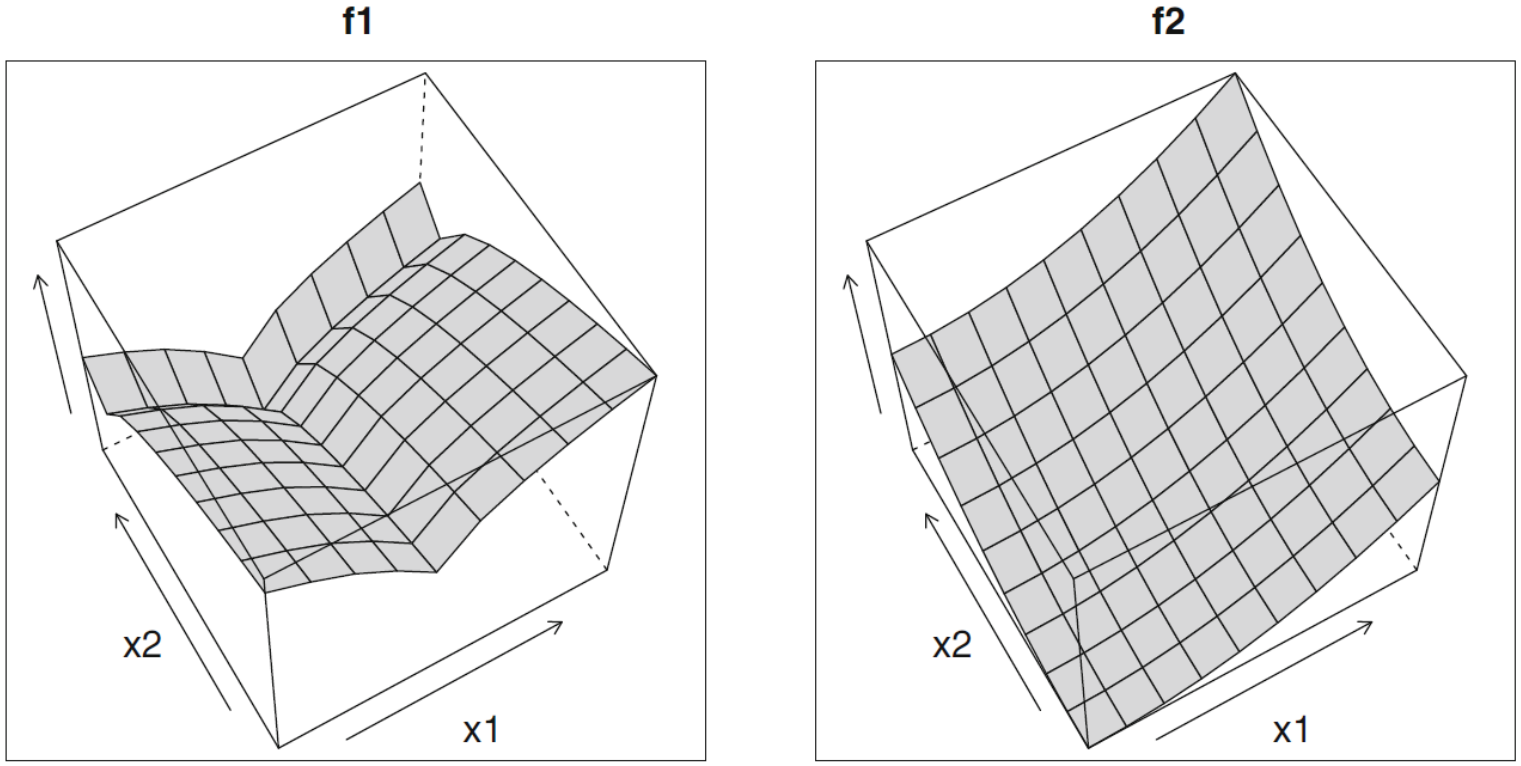





```
rastrigin (D=20)
 EA DE PSO EDA
38 64 101 74   (average best)
100 94 2 58   (%successes)
 EA DE PSO EDA
43674 43830 43722 43646   (average best)
96 100 100 92   (%successes)
```

1.9. Многоцелевая оптимизация

$$\{f_1 = \sum_{i=1}^{D} |x_i - \exp\left((i/D)^2\right)/3|^{0.5},\ f_2 = \sum_{i=1}^{D} (x_i - 0.5\cos(10\pi i/D) - 0.5)^2\}$$



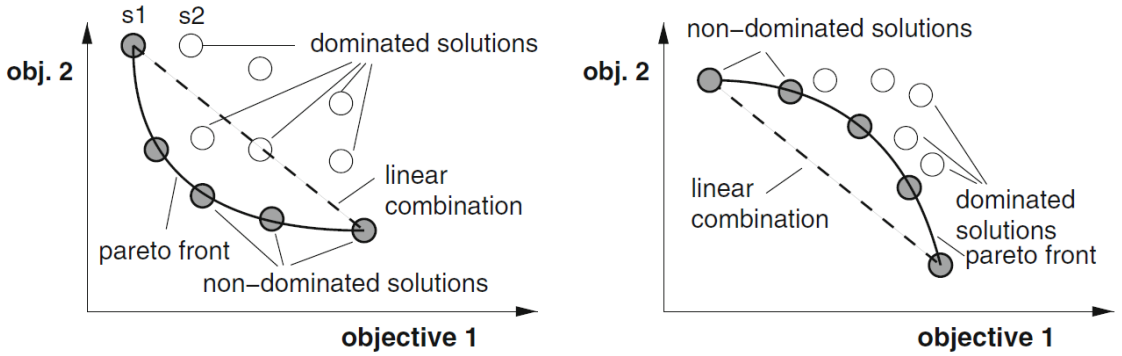
1.9. Многоцелевая оптимизация

ВЗВЕШЕННЫЙ ФУНКЦИОНАЛ

$$Q = w_1 \times g_1 + w_2 \times g_2 + \ldots + w_n \times g_n$$
$$Q = g_1^{w_1} \times g_1^{w_2} \times \ldots \times g_n^{w_n}$$

g_i – цели, w_i – веса.

Недостатки: идеальные веса определить невозможно (основание – интуиция); различные комбинации весов приводят к различным решениям (новые процедуры оптимизации); при корректном определении весов поиск исключает возможные «компромиссы».



1.9. Многоцелевая оптимизация

ЛЕКСИКОГРАФИЧЕСКИЙ ПОДХОД

- У различных целей разный приоритет оптимизации

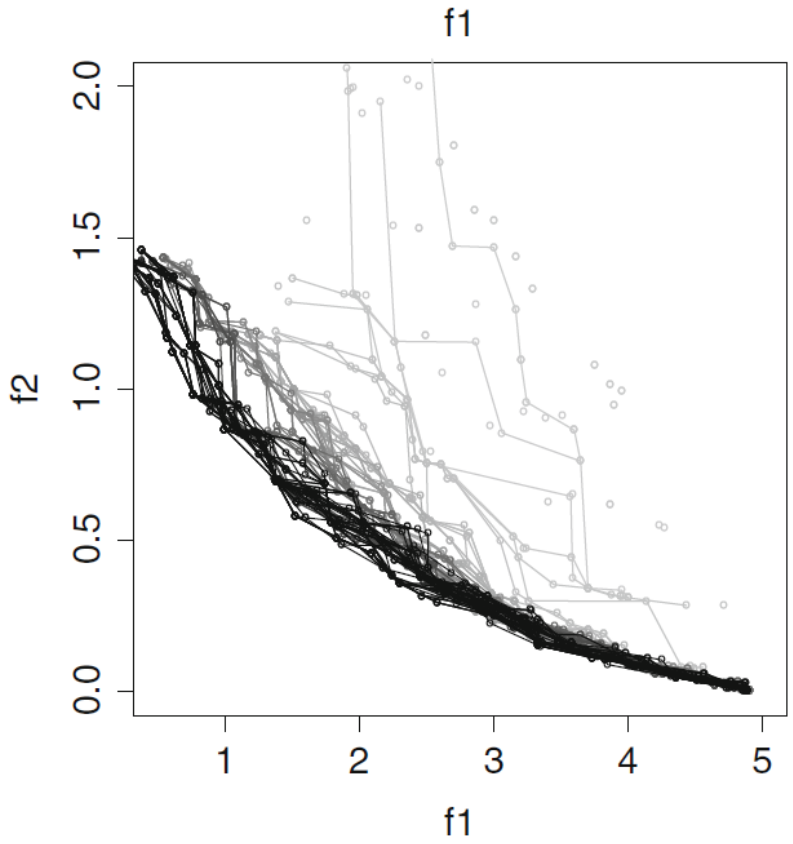
Преимущество: по сравнению с взвешиванием отсутствует проблема смешивания несравнимых показателей.

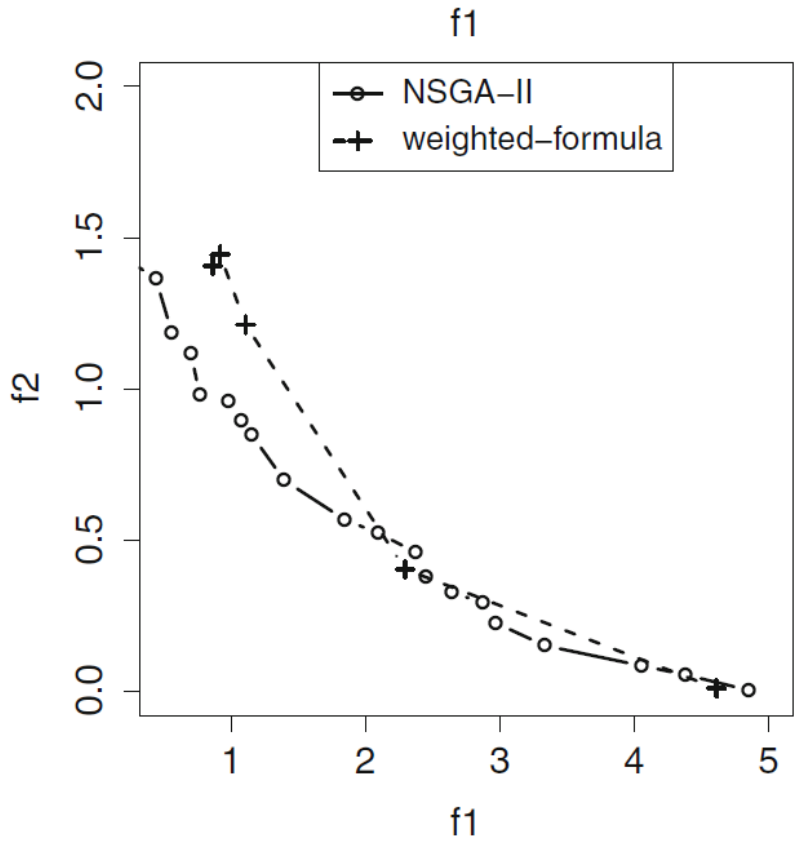
Недостаток: необходимость определения приоритета и порога значимости.

ИСПОЛЬЗОВАНИЕ ПРИНЦИПА ПАРЕТО

Решение s_1 доминирует (в смысле Парето) над решением s_2, если s_1 лучше хотя бы для одного показателя и не хуже, чем s_2 для других. Решение s_i не доминирует, если нет такого s_j, которое доминирует над s_i и линия Парето содержит все недоминирующие решения. Используя такой подход, многоцелевая оптимизация по Парето дает набор недоминирующих решений, а не одно решение.

1.9. Многоцелевая оптимизация





Литература

P. Cortez Modern Optimization with R // Springer International Publishing Switzerland 2014