Липецкий государственный технический университет

Кафедра прикладной математики

КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ МАТЕМАТИЧЕСКИХ ИССЛЕДОВАНИЙ

Лекция 10

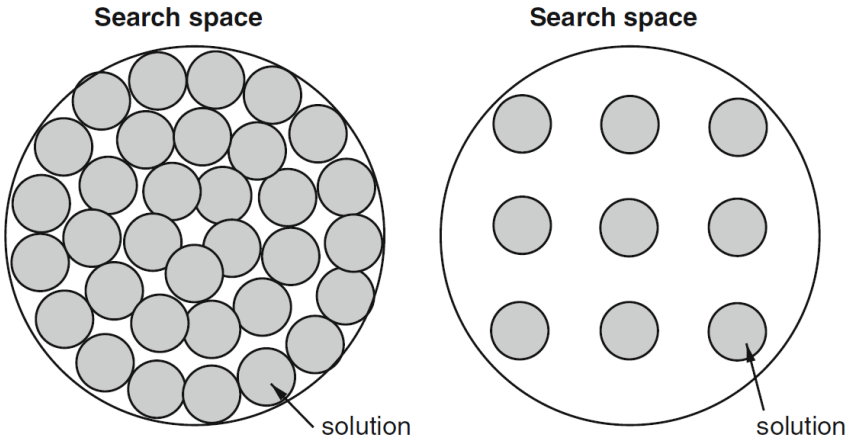
12. Оптимизация в R

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

Липецк - 2022

Outline

12.1. Процедуры слепого поиска



- Другие подходы не дали положительного результата
- Исследуется все пространство решений
- Дискретное пространство решений:
  - Все пространство представляют в виде матрицы и проверяют каждую строку
  - Пространство решений представляют в виде дерева, ветви которого – значения переменных, листья – решения
- Примеры: поиск в глубину, поиск в ширину

- Главный недостаток: процедура невыполнима, если пространство решений непрерывно или очень велико (что часто случается в реальных задачах)
- Жадный алгоритм (жадный поиск) – используется сокращение пространства решений или применяются эвристики
- Метод Монте Карло – генерация случайных точек. Очень популярен, т.к. легок в представлении и в компьютерной реализации

12.1. Процедуры слепого поиска

ПОЛНОСТЬЮ СЛЕПОЙ ПОИСК

`fsearch`

- Пространство поиска должно быть определено как матрица формата `solutions x D (search)`

`dfsearch`

- Рекурсивная процедура поиска в глубину, требует определения области значений для каждой оптимизируемой переменной (`domain`)

Аргументы – оптимизируемые функции, тип оптимизации – минимизация или максимизация, …

```
# full bind search method
#    search - matrix with solutions x D
#    FUN - evaluation function
#    type - "min" or "max"
#    ... - extra parameters for FUN
fsearch=function(search,FUN,type="min",...)
{
 x=apply(search,1,FUN,...) # run FUN over all search rows
 ib=switch(type,min=which.min(x),max=which.max(x))
 return(list(index=ib,sol=search[ib,],eval=x[ib]))
}
```

12.1. Процедуры слепого поиска

ПОЛНОСТЬЮ СЛЕПОЙ ПОИСК

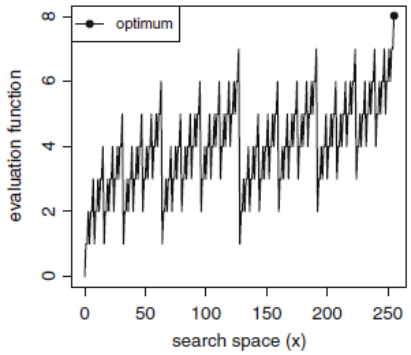
```
# depth-first full search method
#    l - level of the tree
#    b - branch of the tree
#    domain - vector list of size D with domain values
#    FUN - eval function
#    type - "min" or "max"
#    D - dimension (number of variables)
#    x - current solution vector
#    bcur - current best sol
#    ... - extra parameters for FUN
dfsearch=function(l=1,b=1,domain,FUN,type="min",
  D=length(domain),
                x=rep(NA,D),
                bcur=switch(type,min=list(sol=NULL,eval=Inf),
                                 max=list(sol=NULL,eval=-Inf))
                 ...)
{ if((l-1)==D) # "leave" with solution x to be tested:
    { f=FUN(x,...);fb=bcur$eval
      ib=switch(type,min=which.min(c(fb,f)),
                     max=which.max(c(fb,f)))
      if(ib==1) return (bcur) else return(list(sol=x,eval=f))
    }
  else # go through sub branches
    { for(j in 1:length(domain[[l]]))
        { x[l]=domain[[l]][j]
          bcur=dfsearch(l+1,j,domain,FUN,type,D=D,
                        x=x,bcur=bcur,...)
        }
      return(bcur)
    }
}
```

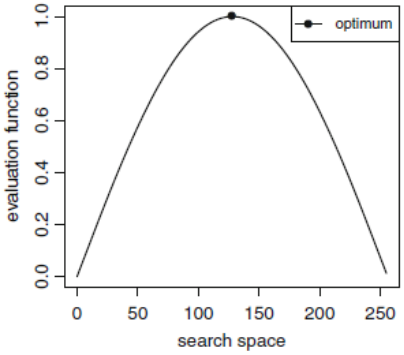
12.1. Процедуры слепого поиска

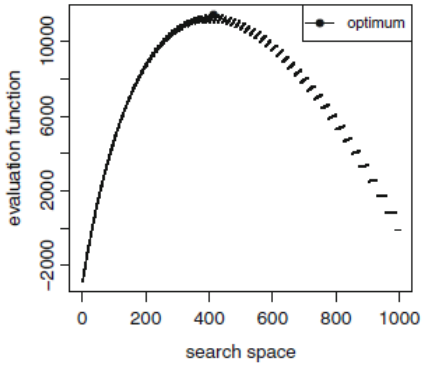
ПРИМЕРЫ

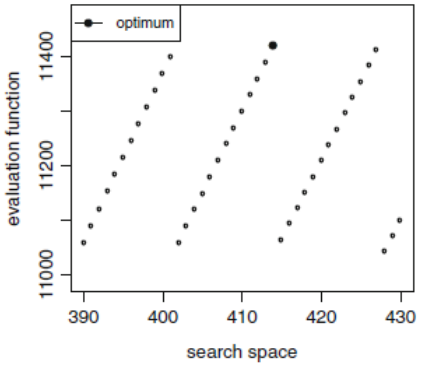
Сумма битов $f_{\text{sum of bits}}(\mathbf{x}) = \sum_{i=1}^{D} x_i \qquad \mathbf{x} = (x_1, \ldots, x_D), (x_i \in \{0, 1\})$

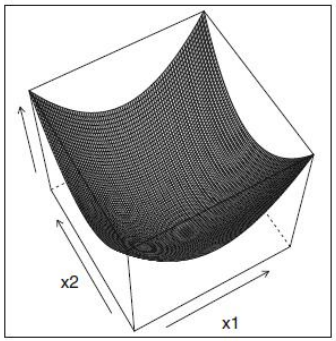
Макс. sin $f_{\max \sin}(\mathbf{x}) = \sin(\pi \frac{x'}{2^D}) \quad x' = \sum_{i=1}^{D} x_i 2^{i-1}$

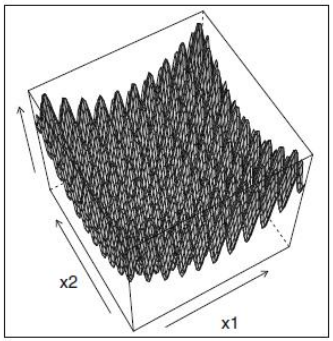












12.1. Процедуры слепого поиска

ПРИМЕРЫ

```
# read D bits from integer x:
binint=function(x,D)
{ x=rev(intToBits(x)[1:D]) # get D bits
  # remove extra 0s from raw type:
  as.numeric(unlist(strsplit(as.character(x),""))[(1:D)*2])
}
# convert binary vector into integer: code inspired in
# http://stackoverflow.com/questions/12892348/
# in-r-how-to-convert-binary-string-to-binary-or-decimal-value
intbin=function(x) sum(2^(which(rev(x==1))-1))
# sum a raw binary object x (evaluation function):
sumbin=function(x) sum(as.numeric(x))
# max sin of binary raw object x (evaluation function):
maxsin=function(x,Dim) sin(pi*(intbin(x))/(2^Dim))
D=8 # number of dimensions
x=0:(2^D-1) # integer search space
# set full search space in solutions x D:
search=t(sapply(x,binint,D=D))
# set the domain values (D binary variables):
domain=vector("list",D)
for(i in 1:D) domain[[i]]=c(0,1) # bits
# sum of bits, fsearch:
S1=fsearch(search,sumbin,"max") # full search
cat("fsearch best s:",S1$sol,"f:",S1$eval,"\n")
# sum of bits, dfsearch:
S2=dfsearch(domain=domain,FUN=sumbin,type="max")
cat("dfsearch best s:",S2$sol,"f:",S2$eval,"\n")
# max sin, fsearch:
S3=fsearch(search,maxsin,"max",Dim=8) # full search
cat("fsearch best s:",S3$sol,"f:",S3$eval,"\n")
# max sin, dfsearch:
S4=dfsearch(domain=domain,FUN=maxsin,type="max",Dim=8)
cat("dfsearch best s:",S4$sol,"f:",S4$eval,"\n")
```

12.1. Процедуры слепого поиска

ПРИМЕРЫ

```
> x=intToBits(7)[1:4]; print(x)
[1] 01 01 01 00
> x=rev(x); print(x)
[1] 00 01 01 01
> x=strsplit(as.character(x),""); print(x)
[[1]]
[1] "0" "0"

[[2]]
[1] "0" "1"

[[3]]
[1] "0" "1"

[[4]]
[1] "0" "1"

> x=unlist(x); print(x)
[1] "0" "0" "0" "1" "0" "1" "0" "1"
> x=as.numeric(x[(1:4)*2]); print(x)
[1] 0 1 1 1
> source("binary-blind.R")
fsearch best s: 1 1 1 1 1 1 1 1 f: 8
dfsearch best s: 1 1 1 1 1 1 1 1 f: 8
fsearch best s: 1 0 0 0 0 0 0 0 f: 1
dfsearch best s: 1 0 0 0 0 0 0 0 f: 1
```

12.1. Процедуры слепого поиска

ЖАДНЫЙ ПОИСК

- Сокращение размерности пространства решений:
  - равномерный план
  - гнездовой план (не полностью слепой, решение по результату)
- Проклятие размерности (оценка сложности $O(L^D)$)
- Дополнительные параметры, которые требуют установки (шаг сетки, количество узлов гнезда)
- Нет гарантии достижения глобального оптимума

```
# standard grid search method (uses fsearch)
#    step - vector with step size for each dimension D
#    lower - vector with lowest values for each dimension
#    upper - vector with highest values for each dimension
#    FUN - evaluation function
#    type - "min" or "max"
#    ... - extra parameters for FUN
```

12.1. Процедуры слепого поиска

ЖАДНЫЙ ПОИСК

```
gsearch=function(step,lower,upper,FUN,type="min",...)
{ D=length(step) # dimension
  domain=vector("list",D) # domain values
  L=vector(length=D) # auxiliary vector
  for(i in 1:D)
     { domain[[i]]=seq(lower[i],upper[i],by=step[i])
       L[i]=length(domain[[i]])
     }
  LS=prod(L)
  s=matrix(ncol=D,nrow=LS) # set the search space
  for(i in 1:D)
    {
     if(i==1) E=1 else E=E*L[i-1]
     s[,i]=rep(domain[[i]],length.out=LS,each=E)
    }
  fsearch(s,FUN,type,...) # best solution
}

# standard grid search method (uses dfsearch)
gsearch2=function(step,lower,upper,FUN,type="min",...)
{ D=length(step) # dimension
  domain=vector("list",D) # domain values
  for(i in 1:D) domain[[i]]=seq(lower[i],upper[i],by=step[i])
  dfsearch(domain=domain,FUN=FUN,type=type,...) # solution
}
```

12.1. Процедуры слепого поиска

ЖАДНЫЙ ПОИСК

```
# nested grid search method (uses fsearch)
#    levels - number of nested levels
ngsearch=function(levels,step,lower,upper,FUN,type,...)
{ stop=FALSE;i=1 # auxiliary objects
  bcur=switch(type,min=list(sol=NULL,eval=Inf),
                   max=list(sol=NULL,eval=-Inf))
  while(!stop) # cycle while stopping criteria is not met
  {
     s=gsearch(step,lower,upper,FUN,type,...)
     # if needed, update best current solution:
     if( (type=="min" && s$eval<bcur$eval)||
         (type=="max" && s$eval>bcur$eval)) bcur=s
     if(i<levels) # update step, lower and upper:
       { step=step/2
         interval=(upper-lower)/4
         lower=sapply(lower,max,s$sol-interval)
         upper=sapply(upper,min,s$sol+interval)
       }
    if(i>=levels || sum((upper-lower)<=step)>0) stop=TRUE
    else i=i+1
  }
  return(bcur) # best solution
}
```

12.1. Процедуры слепого поиска

ПРИМЕРЫ

Цена портфеля

$$f_{\text{bag prices}} = \sum_{i=1}^{D} x_i \times sales(x_i) - cost(x_i)$$

$$cost(x_i) = 100 + u_i \times sales(x_i)$$
$$sales(x_i) = round((1000/\ln(x_i + 200) - 141) \times m_i)$$
$$\mathbf{m} = (2.0, 1.75, 1.5, 1.25, 1.0)$$
$$\mathbf{u} = (\$30, \$25, \$20, \$15, \$10)$$

Сфера

$$f_{\text{sphere}}(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$$

Функция Растригина

$$f_{\text{rastrigin}}(\mathbf{x}) = \sum_{i=1}^{D} (x_i^2 - 10\cos 2\pi x_i + 10)$$

```
# grid search for all bag prices, step of 100$
PTM=proc.time() # start clock
S1=gsearch(rep(100,5),rep(1,5),rep(1000,5),profit,"max")
sec=(proc.time()-PTM)[3] # get seconds elapsed
cat("gsearch best s:",S1$sol,"f:",S1$eval,"time:",sec,"s\n")

# grid search 2 for all bag prices, step of 100$
PTM=proc.time() # start clock
S2=gsearch2(rep(100,5),rep(1,5),rep(1000,5),profit,"max")
sec=(proc.time()-PTM)[3] # get seconds elapsed
cat("gsearch2 best s:",S2$sol,"f:",S2$eval,"time:",sec,"s\n")
```

12.1. Процедуры слепого поиска

ПРИМЕРЫ

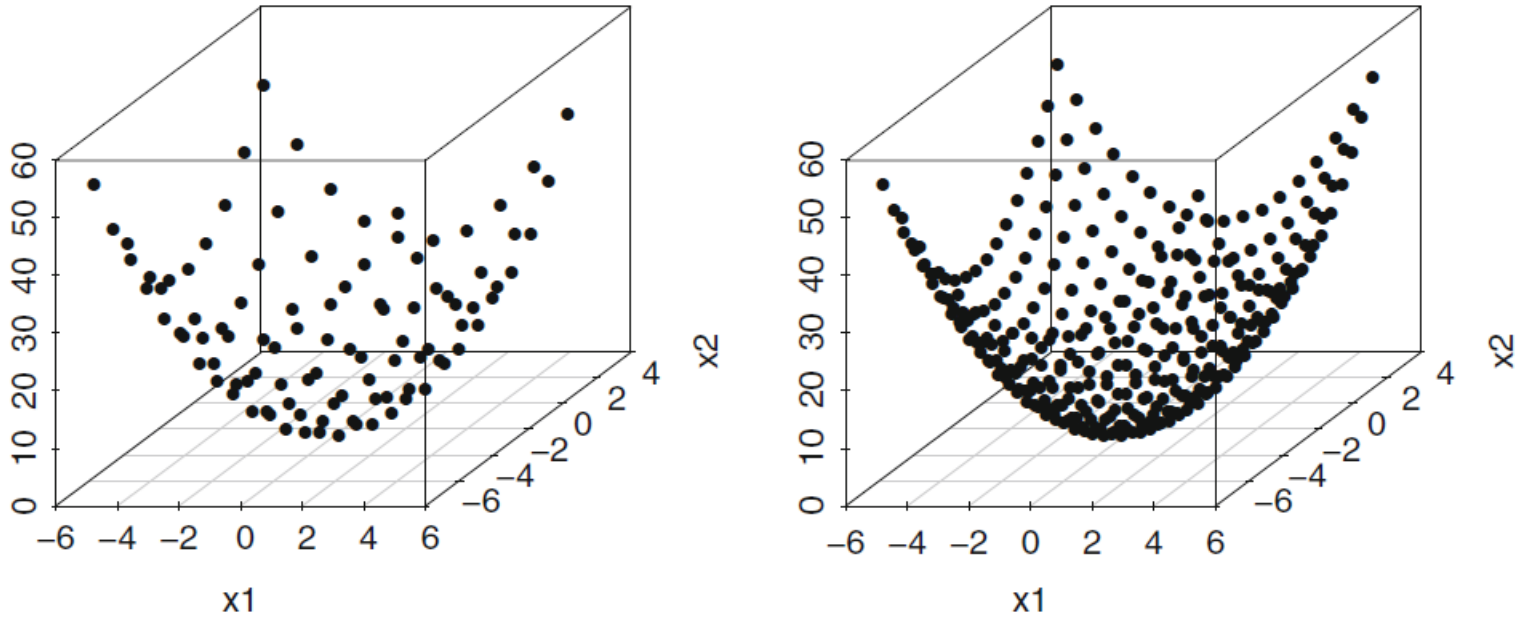
```
# nested grid with 3 levels and initial step of 500$
PTM=proc.time() # start clock
S3=ngsearch(3,rep(500,5),rep(1,5),rep(1000,5),profit,"max")
sec=(proc.time()-PTM)[3] # get seconds elapsed
cat("ngsearch best s:",S3$sol,"f:",S3$eval,"time:",sec,"s\n")
```

```
gsearch best s: 401 401 401 401 501 f: 43142 time: 4.149 s
gsearch2 best s: 401 401 401 401 501 f: 43142 time: 5.654 s
ngsearch best s: 376.375 376.375 376.375 501.375 501.375 f:
    42823 time: 0.005 s
```

```
# real-value functions: sphere and rastrigin:
sphere=function(x) sum(x^2)
rastrigin=function(x) 10*length(x)+sum(x^2-10*cos(2*pi*x))

cat("sphere:\n") # D=2, easy task
S=gsearch(rep(1.1,2),rep(-5.2,2),rep(5.2,2),sphere,"min")
cat("gsearch s:",S$sol,"f:",S$eval,"\n")
S=ngsearch(3,rep(3,2),rep(-5.2,2),rep(5.2,2),sphere,"min")
cat("ngsearch s:",S$sol,"f:",S$eval,"\n")

cat("rastrigin:\n") # D=2, easy task
S=gsearch(rep(1.1,2),rep(-5.2,2),rep(5.2,2),rastrigin,"min")
cat("gsearch s:",S$sol,"f:",S$eval,"\n")
S=ngsearch(3,rep(3,2),rep(-5.2,2),rep(5.2,2),rastrigin,"min")
cat("ngsearch s:",S$sol,"f:",S$eval,"\n")
```

12.1. Процедуры слепого поиска

ПРИМЕРЫ




```
sphere:
gsearch s: 0.3 0.3 f: 0.18
ngsearch s: -0.1 -0.1 f: 0.02
rastrigin:
gsearch s: -1.9 -1.9 f: 11.03966
ngsearch s: -0.1 -0.1 f: 3.83966
```

12.1. Процедуры слепого поиска

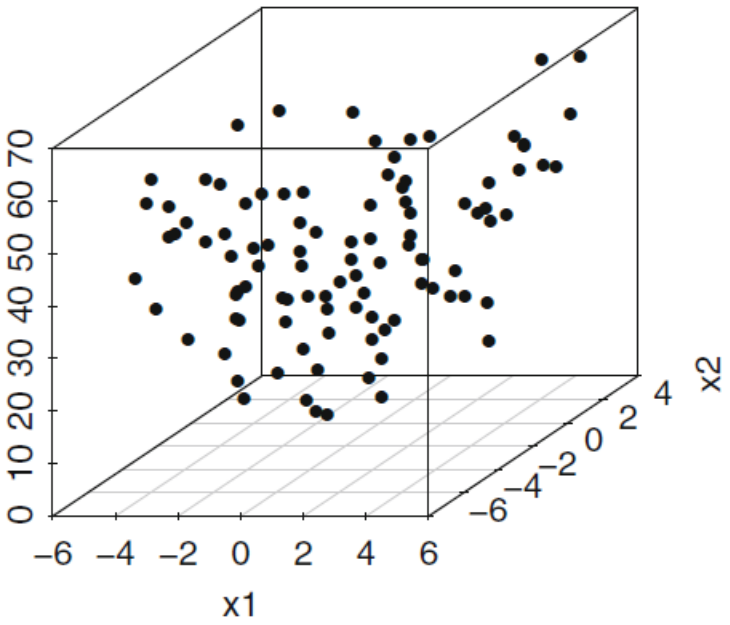
МЕТОДЫ МОНТЕ КАРЛО

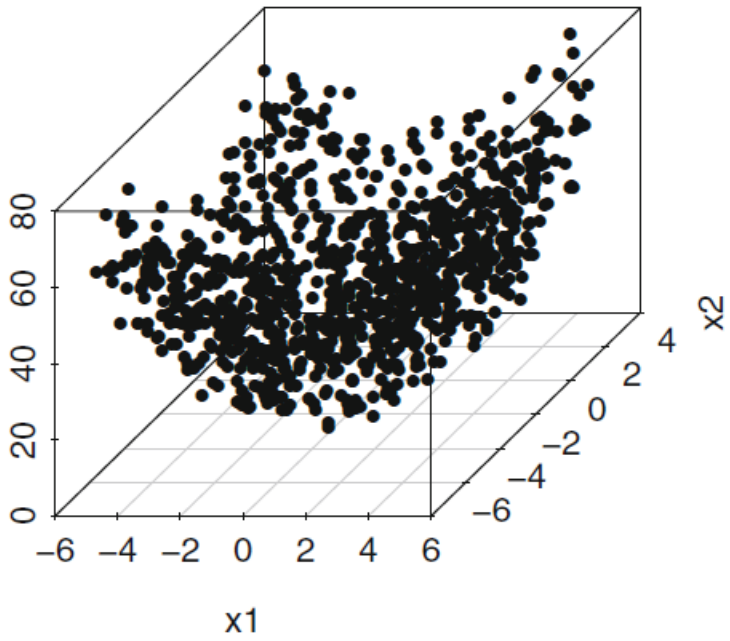
```
# montecarlo uniform search method
#    N - number of samples
#    lower - vector with lowest values for each dimension
#    upper - vector with highest values for each dimension
#    domain - vector list of size D with domain values
#    FUN - evaluation function
#    type - "min" or "max"
#    ... - extra parameters for FUN
mcsearch=function(N,lower,upper,FUN,type="min",...)
{ D=length(lower)
  s=matrix(nrow=N,ncol=D) # set the search space
  for(i in 1:N) s[i,]=runif(D,lower,upper)
  fsearch(s,FUN,type,...) # best solution
}
```

```
D=c(2,30)
label="sphere"
for(i in 1:length(D))
   { S=mcsearch(N,rep(-5.2,D[i]),rep(5.2,D[i]),sphere,"min")
     cat(label,"D:",D[i],"s:",S$sol[1:2],"f:",S$eval,"\n")
   }
label="rastrigin"
for(i in 1:length(D))
   { S=mcsearch(N,rep(-5.2,D[i]),rep(5.2,D[i]),rastrigin,"min")
     cat(label,"D:",D[i],"s:",S$sol[1:2],"f:",S$eval,"\n")
   }
```

12.1. Процедуры слепого поиска

МЕТОДЫ МОНТЕ КАРЛО





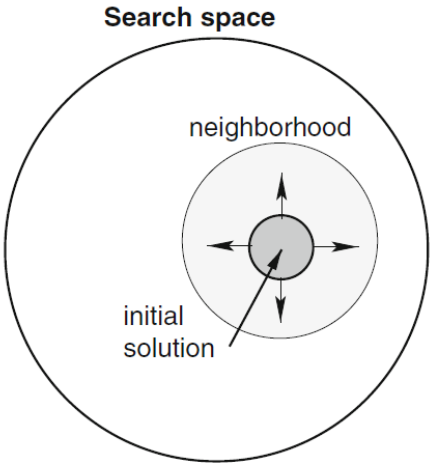

```
monte carlo search (N: 10000 )
bag prices:s: 349.7477 369.1669 396.1959 320.5007 302.3327 f:
    42508
sphere D: 2 s: -0.01755296 0.0350427 f: 0.001536097
sphere D: 30 s: -0.09818928 -1.883463 f: 113.7578
rastrigin D: 2 s: -0.0124561 0.02947438 f: 0.2026272
rastrigin D: 30 s: 0.6508581 -3.043595 f: 347.1969
```

ПОИСК ВОСХОЖДЕНИЕМ К ВЕРШИНЕ

Поиск восхождением к вершине (восхождение) — это техника математической оптимизации, принадлежащая семейству алгоритмов локального поиска. Алгоритм является методом итерации, который начинается с произвольного решения задачи, а затем пытается найти лучшее решение путём пошагового изменения одного из элементов решения. Если решение даёт лучшее решение, делается приращение для получения нового решения и оно делается, пока не достигнем момента, в котором улучшение найти не удаётся.

Algorithm Pure hill climbing optimization method

1: **Inputs:** S, f, C ▷ S is the initial solution, f is the evaluation function, C includes control parameters
2: $i \leftarrow 0$ ▷ i is the number of iterations of the method
3: **while** not $termination\_criteria(S, f, C, i)$ **do**
4: $S' \leftarrow change(S, C)$ ▷ new solution
5: $B \leftarrow best(S, S', f)$ ▷ best solution for next iteration
6: $S \leftarrow B$ ▷ deterministic select function
7: $i \leftarrow i + 1$
8: **end while**
9: **Output:** B ▷ the best solution



12.2. Процедуры локального поиска

ПОИСК ВОСХОЖДЕНИЕМ К ВЕРШИНЕ

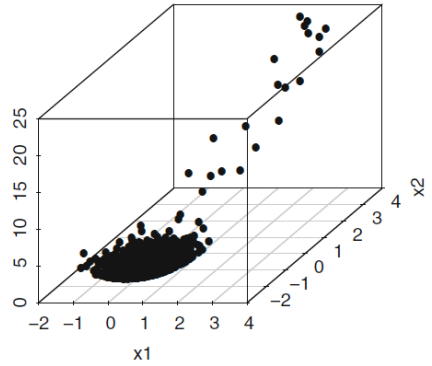
```
# pure hill climbing:
#    par - initial solution
#    fn - evaluation function
#    change - function to generate the next candidate
#    lower - vector with lowest values for each dimension
#    upper - vector with highest values for each dimension
#    control - list with stopping and monitoring method:
#       $maxit - maximum number of iterations
#       $REPORT - frequency of monitoring information
#    type - "min" or "max"
#    ... - extra parameters for FUN
hclimbing=function(par,fn,change,lower,upper,control,
                   type="min",...)
{ fpar=fn(par,...)
  for(i in 1:control$maxit)
     {
      par1=change(par,lower,upper)
      fpar1=fn(par1,...)
      if(control$REPORT>0 &&(i==1||i%%control$REPORT==0))
        cat("i:",i,"s:",par,"f:",fpar,"s'",par1,"f:",fpar1,"\n")
      if(    (type=="min" && fpar1<fpar)
          || (type=="max" && fpar1>fpar)) { par=par1;fpar=fpar1 }
     }
  if(control$REPORT>=1) cat("best:",par,"f:",fpar,"\n")
  return(list(sol=par,eval=fpar))
}
```

12.2. Процедуры локального поиска

ПОИСК ВОСХОЖДЕНИЕМ К ВЕРШИНЕ

```
# slight random change of vector par:
#    par - initial solution
#    lower - vector with lowest values for each dimension
#    upper - vector with highest values for each dimension
#    dist - random distribution function
#    round - use integer (TRUE) or continuous (FALSE) search
#    ... - extra parameters for dist
#    examples: dist=rnorm, mean=0, sd=1; dist=runif, min=0,max=1
hchange=function(par,lower,upper,dist,round=TRUE,...)
{ D=length(par) # dimension
  step=dist(D,...) # slight step
  if(round) step=round(step)
  par1=par+step
  # return par1 within [lower,upper]:
  return(ifelse(par1<lower,lower,ifelse(par1>upper,upper,par1)))
}
```

```
s=runif(D,-5.2,5.2) # initial search
hclimbing(s,sphere,change=rchange,lower=rep(-5.2,D),
          upper=rep(5.2,D),control=C,type="min")
```



12.2. Процедуры локального поиска

ИМИТАЦИЯ ОТЖИГА

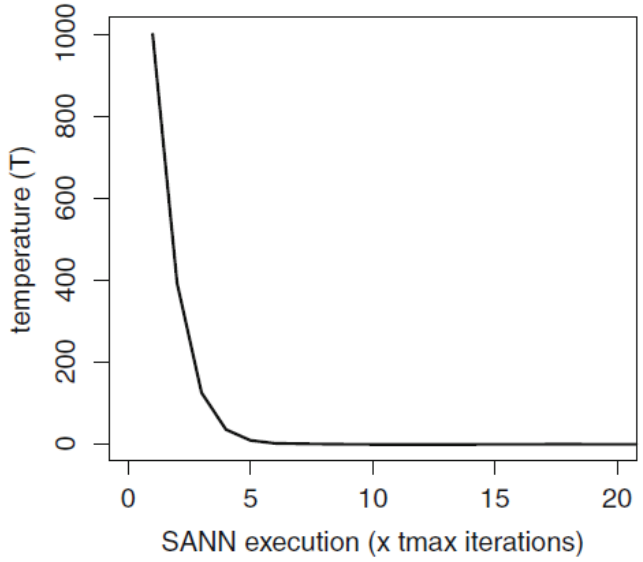
Algorithm Simulated annealing search as implemented by the `optim` function

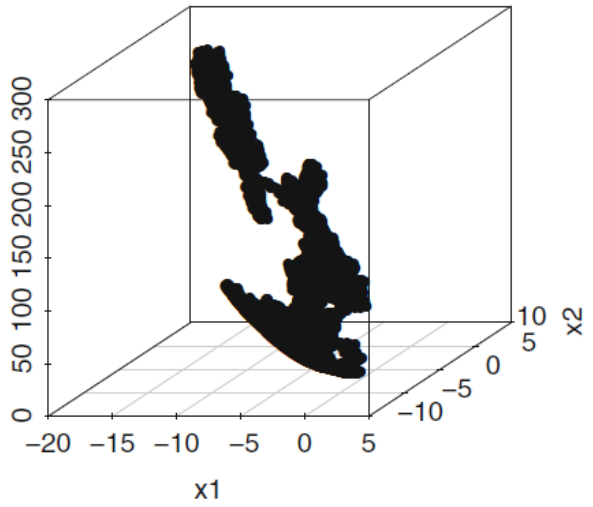
1: **Inputs:** S, f, C ▷ S is the initial solution, f is the evaluation function, C contains control parameters ($maxit$, T and $tmax$)
2: $maxit \leftarrow get\_maxit(C)$ ▷ maximum number of iterations
3: $T \leftarrow get\_temperature(C)$ ▷ temperature, should be a high number
4: $tmax \leftarrow get\_tmax(C)$ ▷ number of evaluations at each temperature
5: $fs \leftarrow f(S)$ ▷ evaluation of S
6: $B \leftarrow S$ ▷ best solution
7: $i \leftarrow 0$ ▷ i is the number of iterations of the method
8: **while** $i < maxit$ **do** ▷ $maxit$ is the termination criterion
9: **for** $j = 1 \rightarrow tmax$ **do** ▷ cycle j from 1 to $tmax$
10: $S' \leftarrow change(S, C)$ ▷ new solution (might depend on T)
11: $fs' \leftarrow f(S')$ ▷ evaluation of S'
12: $r \leftarrow \mathscr{U}(0, 1)$ ▷ random number, uniform within [0, 1]
13: $p \leftarrow \exp\left(\frac{fs' - fs}{T}\right)$ ▷ probability $P(S, S', T)$ (Metropolis function)
14: **if** $fs' < fs \vee r < p$ **then** $S \leftarrow S'$ ▷ accept best solution or worst if $r < p$
15: **end if**
16: **if** $fs' < fs$ **then** $B \leftarrow S'$
17: **end if**
18: $i \leftarrow i + 1$
19: **end for**
20: $T \leftarrow \frac{T}{\log(i/tmax) \times tmax + \exp(1)}$ ▷ cooling step (decrease temperature)
21: **end while**
22: **Output:** B ▷ the best solution

12.2. Процедуры локального поиска

ИМИТАЦИЯ ОТЖИГА

Алгоритм основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Предполагается, что атомы уже выстроились в кристаллическую решётку, но ещё допустимы переходы отдельных атомов из одной ячейки в другую. Предполагается, что процесс протекает при постепенно понижающейся температуре. Переход атома из одной ячейки в другую происходит с некоторой вероятностью, причём вероятность уменьшается с понижением температуры. Устойчивая кристаллическая решётка соответствует минимуму энергии атомов, поэтому атом либо переходит в состояние с меньшим уровнем энергии, либо остаётся на месте.





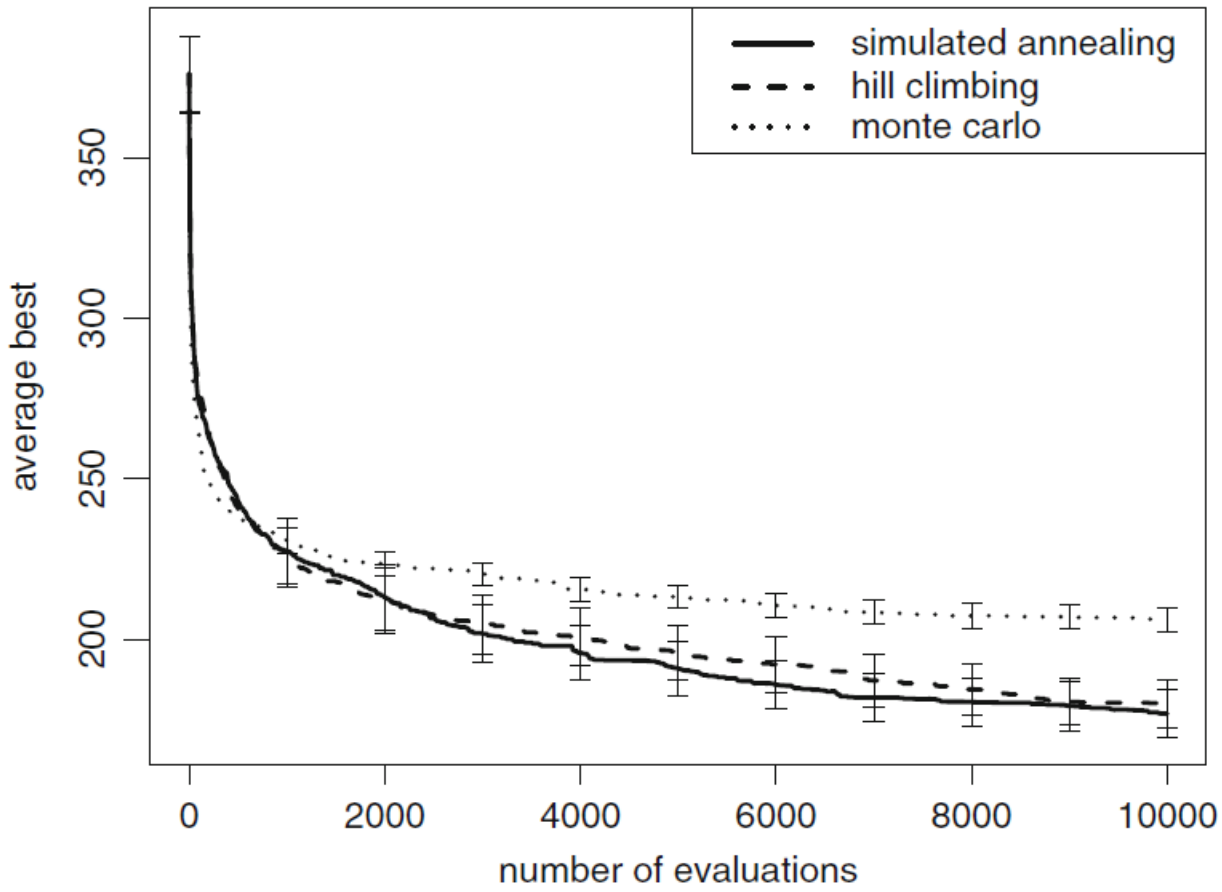
12.2. Процедуры локального поиска

ПОИСК С ЗАПРЕТАМИ

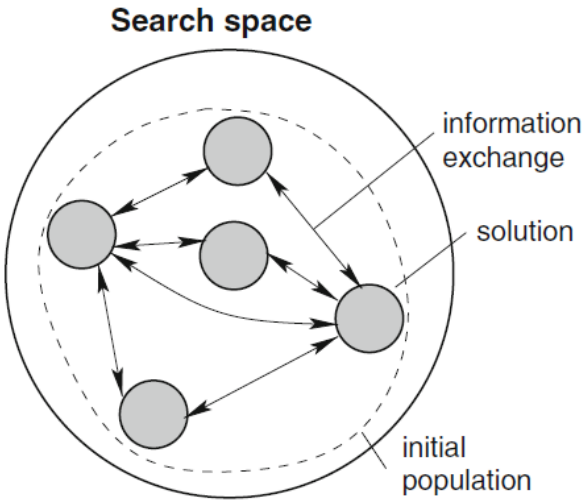
Algorithm Tabu search

1: **Inputs:** S, f, C ▷ S is the initial solution, f is the evaluation function, C contains control parameters ($maxit$, L and N)
2: $maxit \leftarrow get\_maxit(C)$ ▷ maximum number of iterations
3: $L \leftarrow get\_L(C)$ ▷ length of the tabu list
4: $N \leftarrow get\_N(C)$ ▷ number of neighbor configurations to check at each iteration
5: $List \leftarrow \{\}$ ▷ tabu list (first in, first-out queue)
6: $i \leftarrow 0$ ▷ i is the number of iterations of the method
7: **while** $i < maxit$ **do** ▷ $maxit$ is the termination criterion
8: **for** $j = 1 \rightarrow N$ **do** ▷ cycle j from 1 to N
9: $S' \leftarrow change(S, C)$ ▷ new solution
10: $CList \leftarrow \{\}$ ▷ candidate list
11: **if** $S' \notin List$ **then** $CList \leftarrow CList \cup S'$ ▷ add S' into $CList$
12: **end if**
13: **end for**
14: $S' \leftarrow best(CList, f)$ ▷ get best candidate solution
15: **if** $isbest(S', S, f)$ **then** ▷ if S' is better than S
16: $List \leftarrow List \cup S'$ ▷ enqueue S' into $List$
17: **if** $length(List) > L$ **then** $dequeue(L)$ ▷ remove oldest element
18: **end if**
19: $S \leftarrow S'$ ▷ set S as the best solution S'
20: **end if**
21: $i \leftarrow i + 1$
22: **end while**
23: **Output:** S ▷ the best solution

12.2. Процедуры локального поиска



12.3. 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.

12.4. 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)

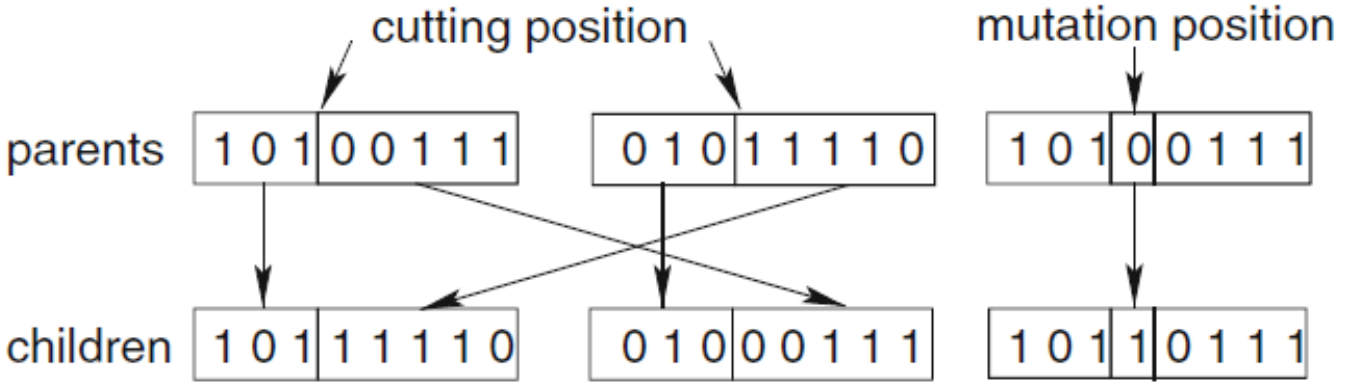
Размножение – совокупность кроссовера и мутаций. Участвуют два предка, получается потомок, отличный от остальных из-за мутации.

12.4. Генетические и эволюционные алгоритмы

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

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

12.4. Генетические и эволюционные алгоритмы



Кроссовер **мутация**

Мутация «реальных» генов

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

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

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

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

12.4. Генетические и эволюционные алгоритмы

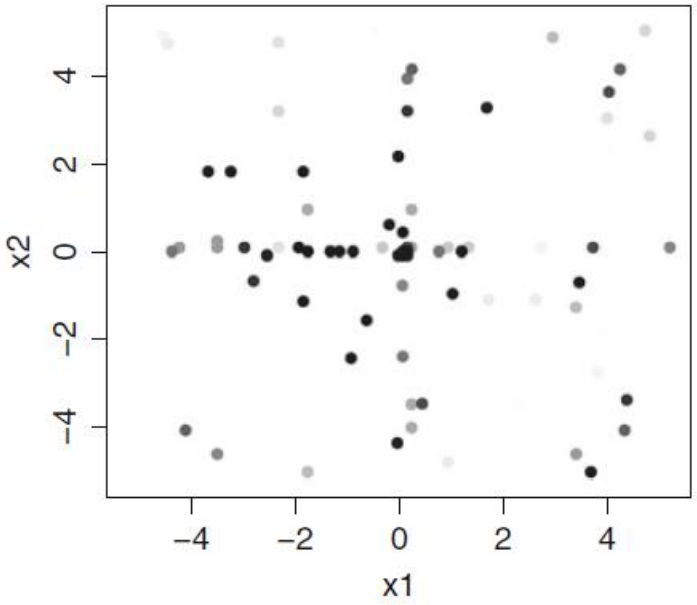
```
### bag-genalg.R file ###
library(genalg) # load genalg package
source("functions.R") # load the profit function

# 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)
}
```

12.4. 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
```

12.4. 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
```

12.5. Дифференциальная эволюция

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

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

- 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.

12.5. 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

12.5. 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
```

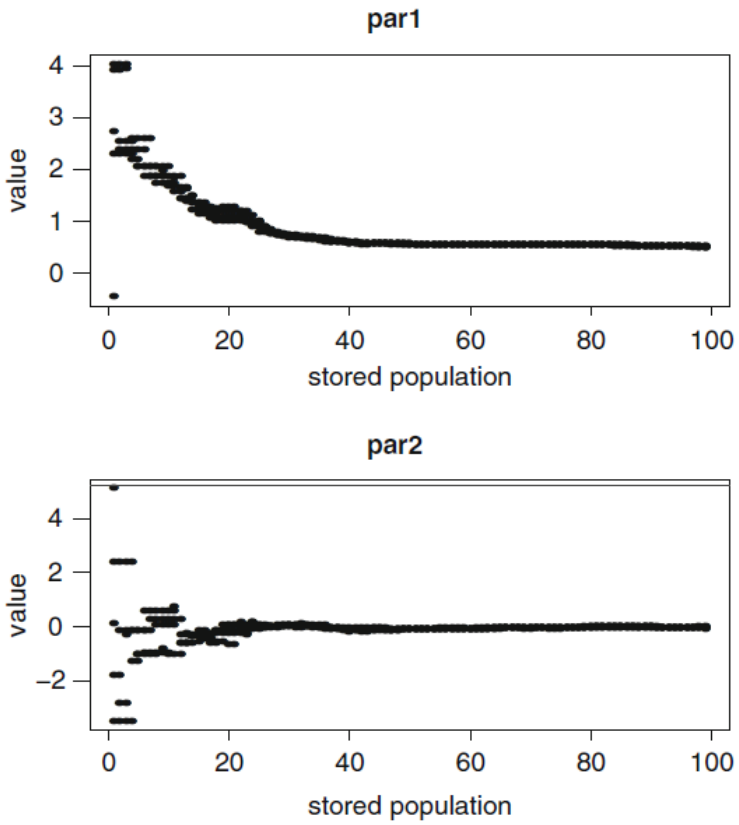
12.6. Роиные алгоритмы

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

12.6. 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")
```



12.7. Estimation of Distribution Algorithm

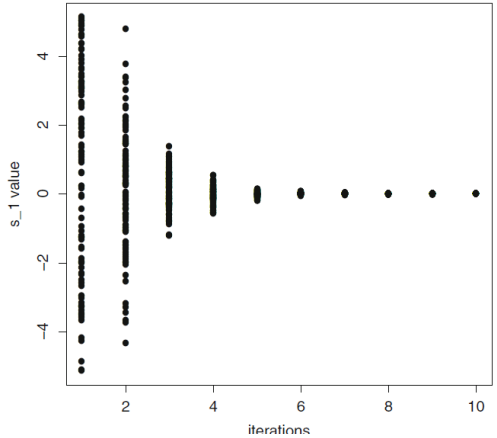
Generic EDA pseudo-code implemented in `copulaedas` package, adapted from Gonzalez-Fernandez and Soto (2012)

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

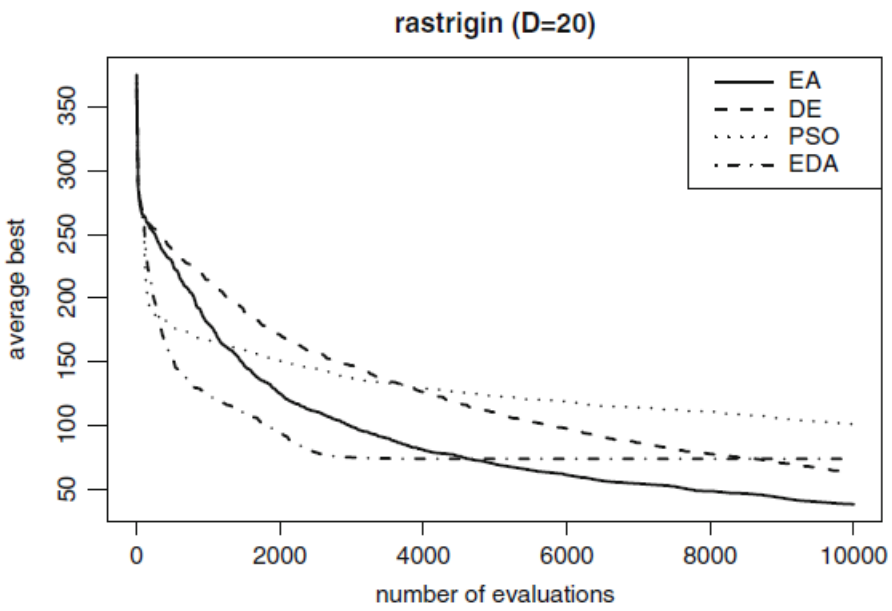
12.7. Estimation of Distribution Algorithm

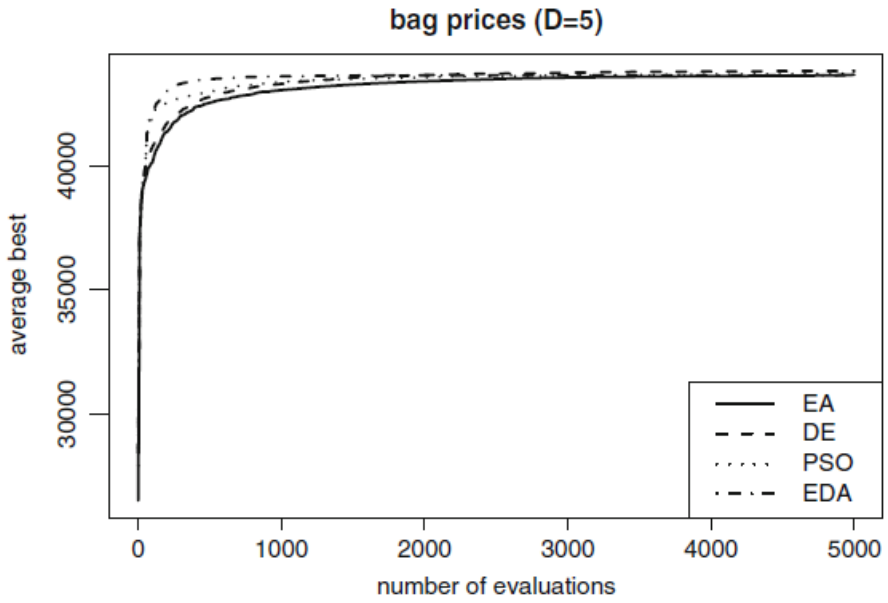
```
### sphere-EDA.R file ###
library(copulaedas)
sphere=function(x) sum(x^2)
D=2; maxit=10; LP=5
set.seed(12345) # set for replicability
# set termination criterion and report method:
setMethod("edaTerminate","EDA",edaTerminateMaxGen)
setMethod("edaReport","EDA",edaReportSimple)
# set EDA type:
UMDA=CEDA(copula="indep",margin="norm",popSize=LP,maxGen=maxit)
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
```



12.12. Сравнение популяционных алгоритмов

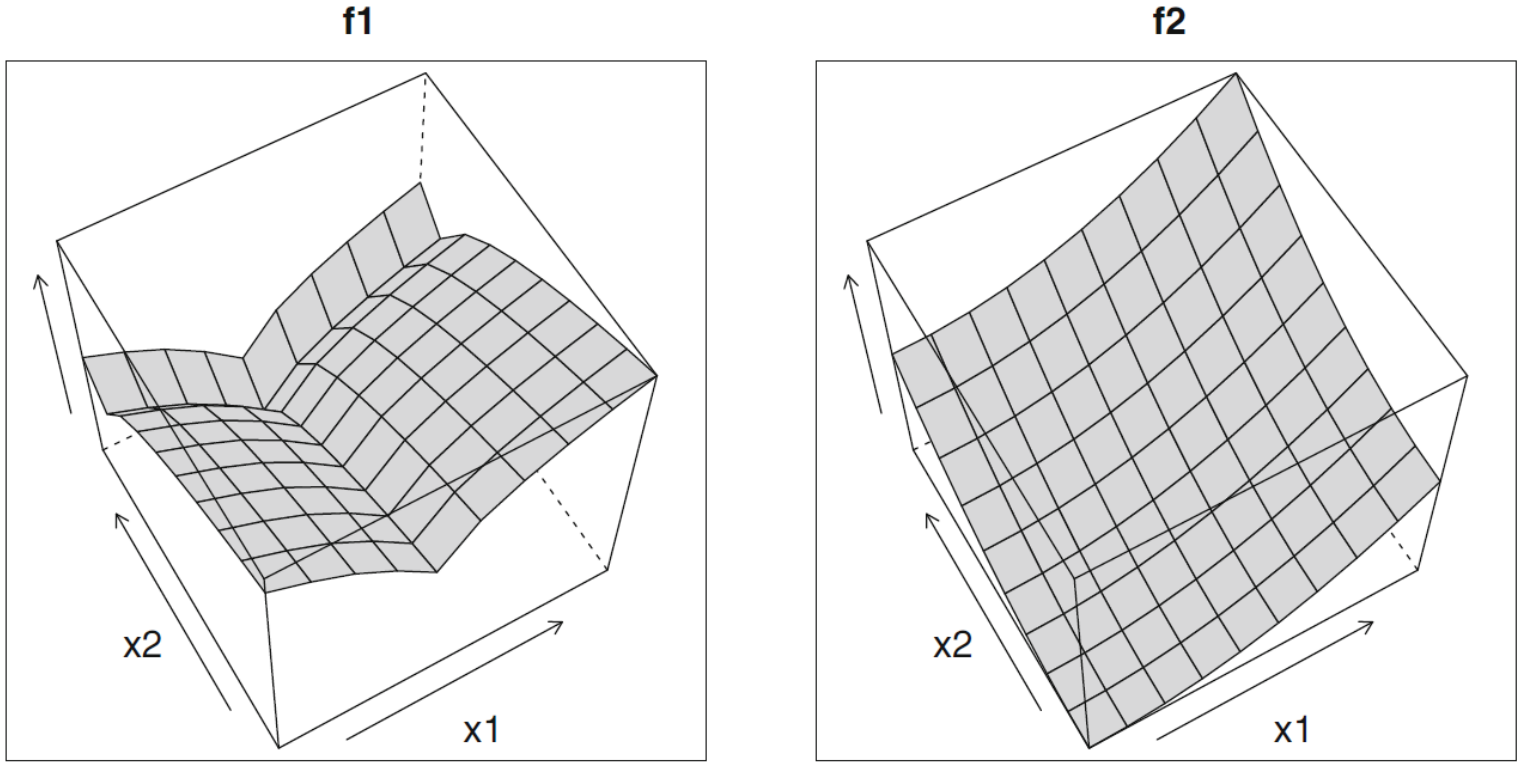





```
rastrigin (D=20)
 EA DE PSO EDA
38 64 101 74   (average best)
100 94 2 58   (%successes)
 EA DE PSO EDA
43674 43830 43722 43646   (average best)
96 100 100 92   (%successes)
```

12.9. Многоцелевая оптимизация

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



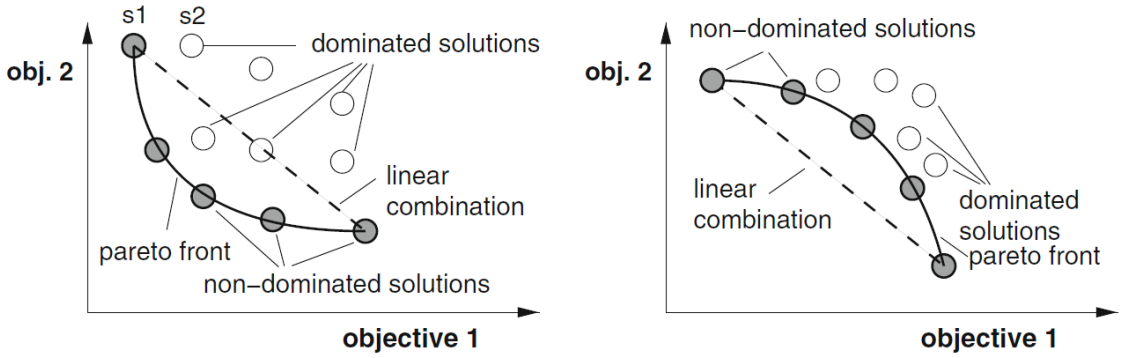
12.9. Многоцелевая оптимизация

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».



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.

Литература

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