Липецкий государственный технический университет

Кафедра прикладной математики

КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ МАТЕМАТИЧЕСКИХ ИССЛЕДОВАНИЙ

Лекция 1

Оптимизация в R

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

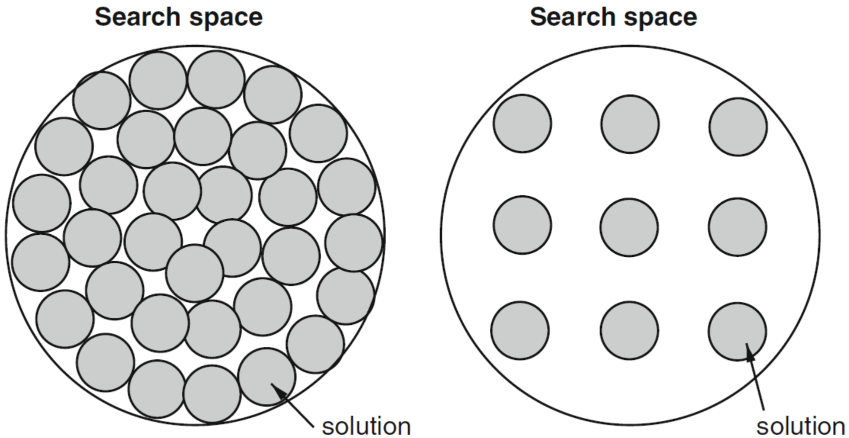
Липецк - 2021

Outline

1.1. Процедуры слепого поиска

1.2. Процедуры локального поиска

1.1. 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

1.1. 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]))
}
```

ПОЛНОСТЬЮ СЛЕПОЙ ПОИСК

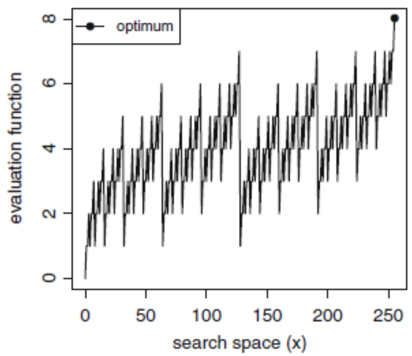
```
# depth-first full search method
#    l - level of the tree
#    b - branch of the tree
#    domain - vector list of size D with domain values
#    FUN - eval function
#    type - "min" or "max"
#    D - dimension (number of variables)
#    x - current solution vector
#    bcur - current best sol
#    ... - 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)
     }
}
```

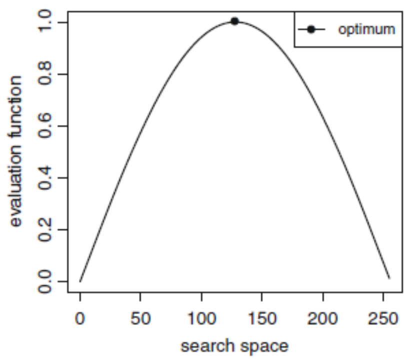
1.1. Процедуры слепого поиска

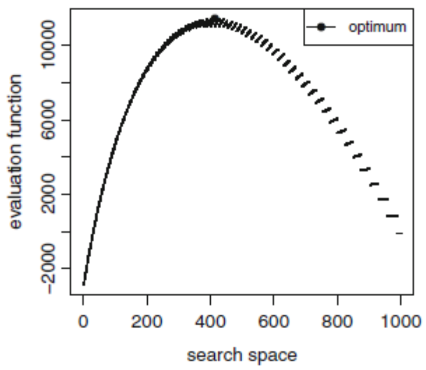
ПРИМЕРЫ

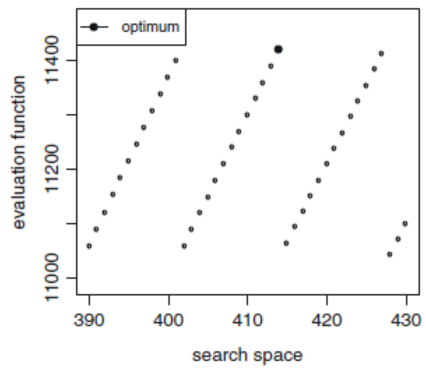
Сумма битов $\quad f_{\text{sum of bits}}(\mathbf{x}) = \sum_{i=1}^{D} x_i \qquad \mathbf{x} = (x_1, \ldots, x_D), \ (x_i \in \{0, 1\})$

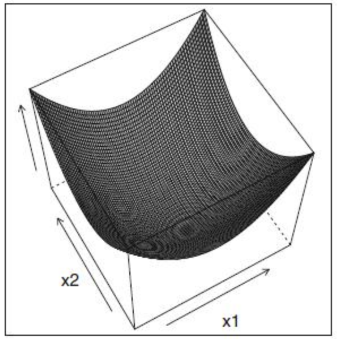
Макс. sin $\quad f_{\max \sin}(\mathbf{x}) = \sin\left(\pi \frac{x'}{2^D}\right) \qquad x' = \sum_{i=1}^{D} x_i 2^{i-1}$

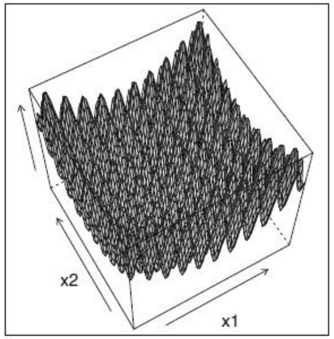












1.1. Процедуры слепого поиска

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")
```

1.1. 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
```

1.1. 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
```

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
}
```

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
}
```

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")
```

1.1. 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")
```

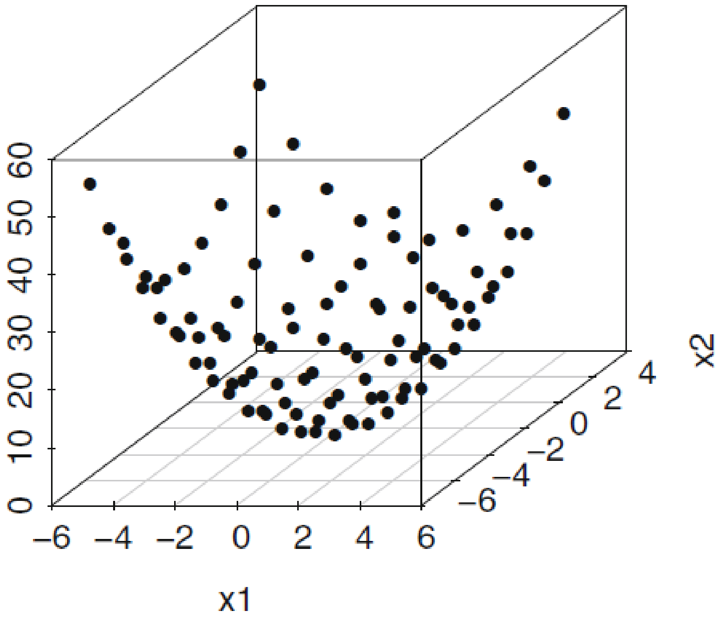
```
gsearch best s: 401 401 401 401 501 f: 43142 time: 4.149 s
gsearch2 best s: 401 401 401 401 501 f: 43142 time: 5.654 s
ngsearch best s: 376.375 376.375 376.375 501.375 501.375 f:
    42823 time: 0.005 s
```

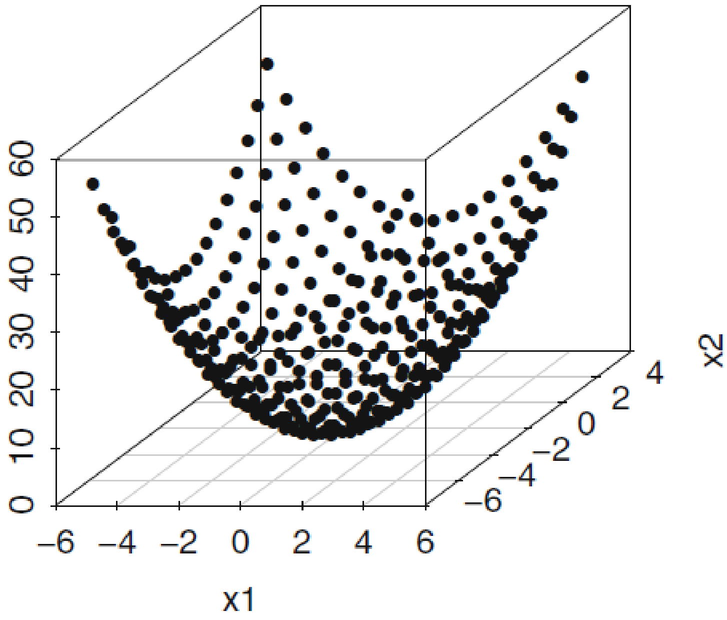
```
# real-value functions: sphere and rastrigin:
sphere=function(x) sum(x^2)
rastrigin=function(x) 10*length(x)+sum(x^2-10*cos(2*pi*x))

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")
```

ПРИМЕРЫ





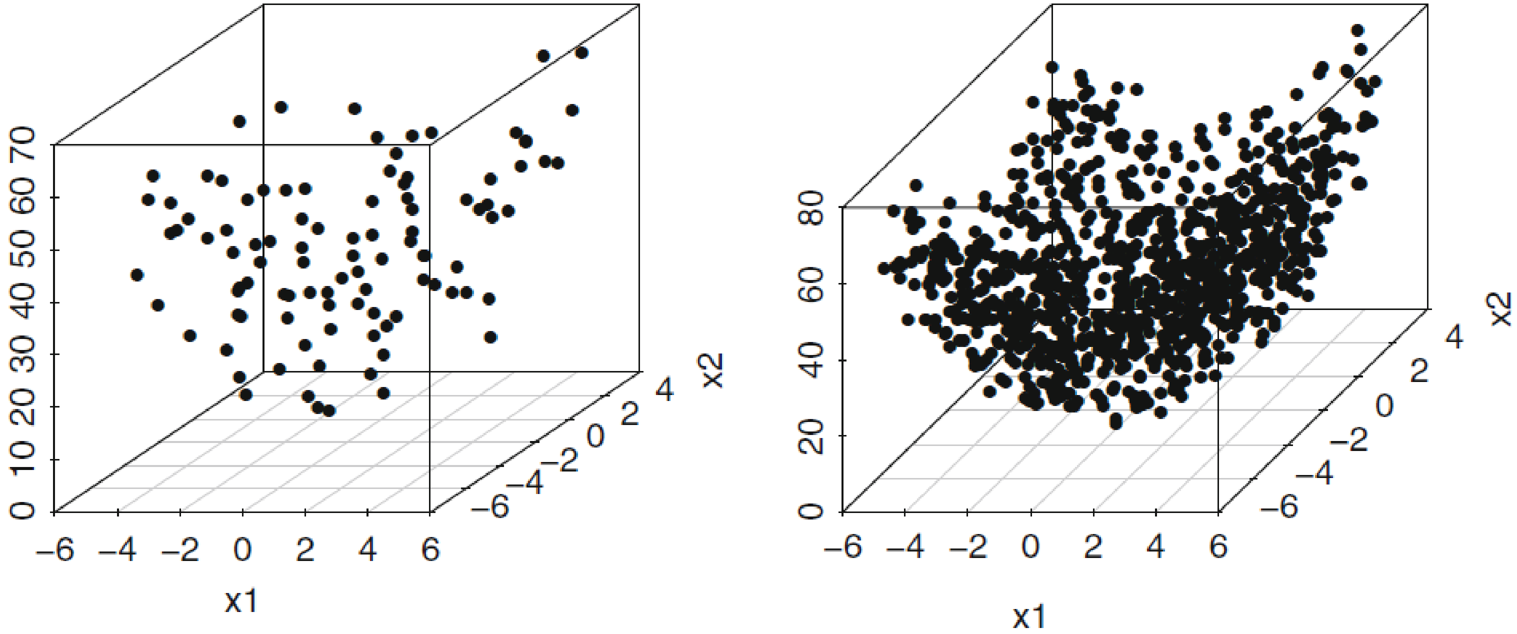

```
sphere:
gsearch s: 0.3 0.3 f: 0.18
ngsearch s: -0.1 -0.1 f: 0.02
rastrigin:
gsearch s: -1.9 -1.9 f: 11.03966
ngsearch s: -0.1 -0.1 f: 3.83966
```

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")
   }
```

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
```

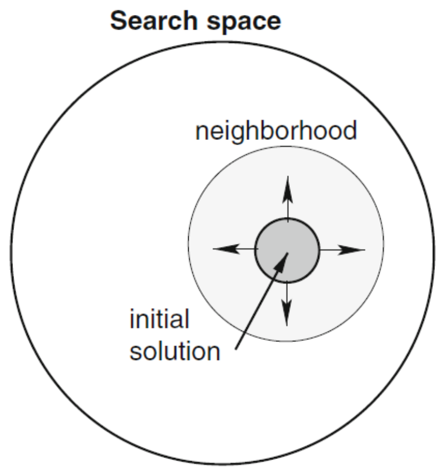
1.2. Процедуры локального поиска

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

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



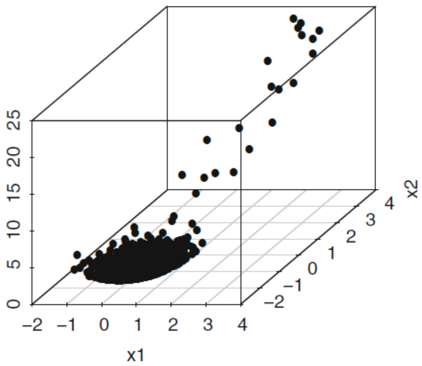
ПОИСК ВОСХОЖДЕНИЕМ К ВЕРШИНЕ

```
# 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))
}
```

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")
```



ИМИТАЦИЯ ОТЖИГА

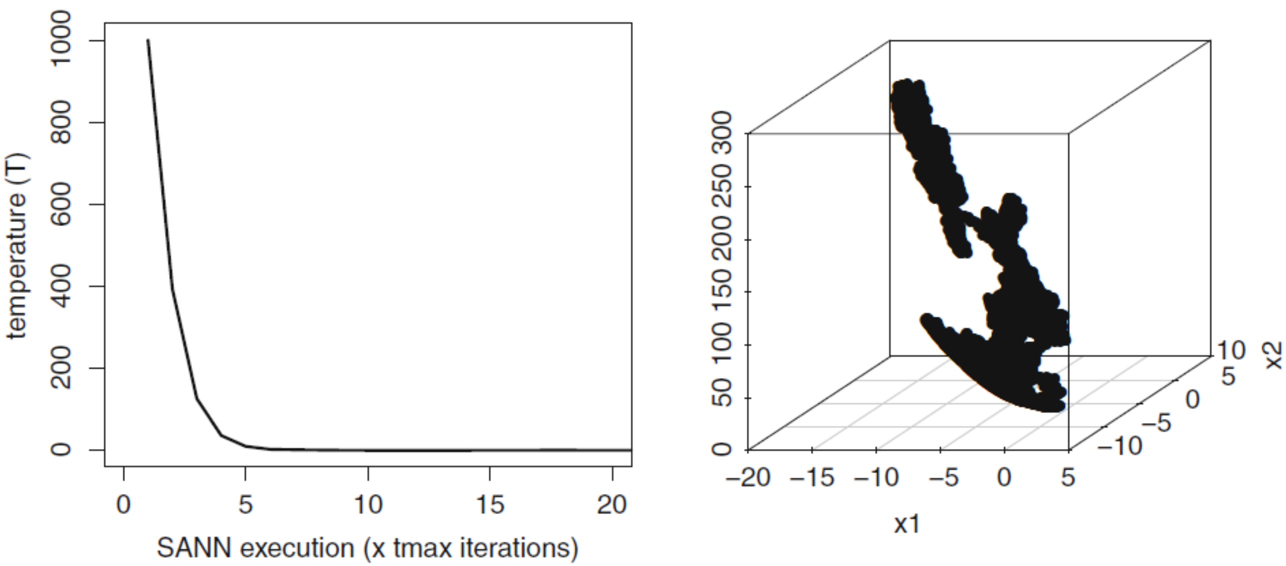
Algorithm Simulated annealing search as implemented by the `optim` function

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

1.2. Процедуры локального поиска

ИМИТАЦИЯ ОТЖИГА

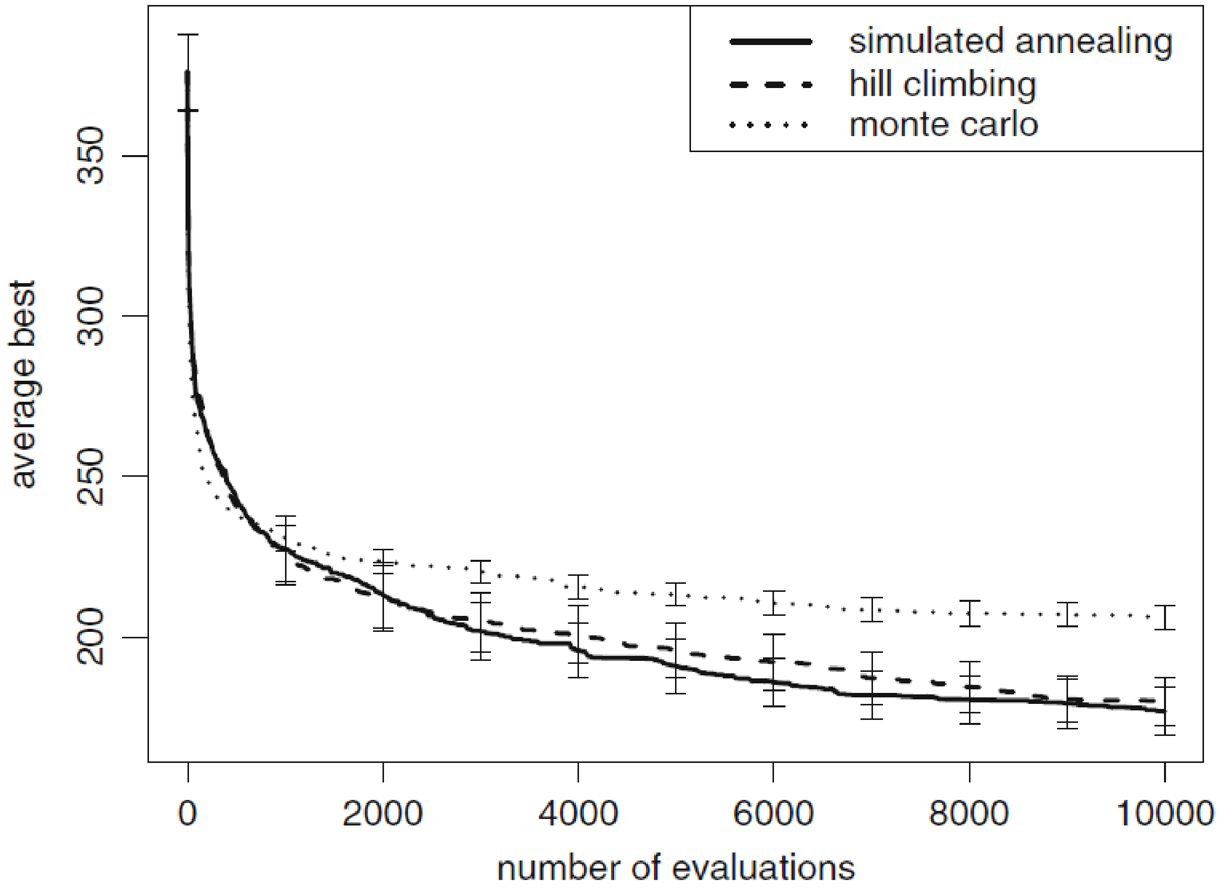
Алгоритм основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Предполагается, что атомы уже выстроились в кристаллическую решётку, но ещё допустимы переходы отдельных атомов из одной ячейки в другую. Предполагается, что процесс 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.



ПОИСК С ЗАПРЕТАМИ

Algorithm Tabu search

```
Inputs: S, f, C    ▷ S is the initial solution, f is the evaluation function, C contains control parameters (maxit, L and N)
maxit ← get_maxit(C)    ▷ maximum number of iterations
L ← get_L(C)    ▷ length of the tabu list
N ← get_N(C)    ▷ number of neighbor configurations to check at each iteration
List ← {}    ▷ tabu list (first in, first-out queue)
i ← 0    ▷ i is the number of iterations of the method
while i < maxit do    ▷ maxit is the termination criterion
    for j = 1 → N do    ▷ cycle j from 1 to N
        S′ ← change(S, C)    ▷ new solution
        CList ← {}    ▷ candidate list
        if S′ ∉ List then CList ← CList ∪ S′    ▷ add S′ into CList
        end if
    end for
    S′ ← best(CList, f)    ▷ get best candidate solution
    if isbest(S′, S, f) then    ▷ if S′ is better than S
        List ← List ∪ S′    ▷ enqueue S′ into List
        if length(List) > L then dequeue(L)    ▷ remove oldest element
        end if
        S ← S′    ▷ set S as the best solution S′
    end if
    i ← i + 1
end while
Output: S    ▷ the best solution
```


simulated annealing
hill climbing
monte carlo
average best
350
300
250
200
0
2000
4000
6000
8000
10000
number of evaluations

Литература

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