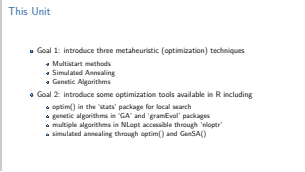
Metaheuristics and Optimization in R

DS 775

This Unit

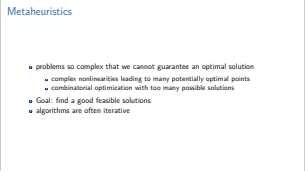
- Goal 1: introduce three metaheuristic (optimization) techniques
  - Multistart methods
  - Simulated Annealing
  - Genetic Algorithms
- Goal 2: introduce some optimization tools available in R including
  - optim() in the 'stats' package for local search
  - genetic algorithms in 'GA' and 'gramEvol' packages
  - multiple algorithms in NLopt accessible through 'nloptr'
  - simulated annealing through optim() and GenSA()



- goes with audio01.mp3

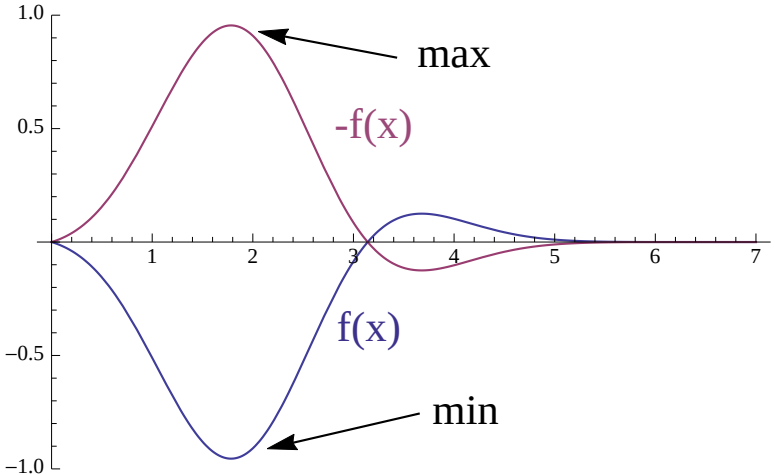
Metaheuristics

- problems so complex that we cannot guarantee an optimal solution
  - complex nonlinearities leading to many potentially optimal points
  - combinatorial optimization with too many possible solutions
- Goal: find a good feasible solutions
- algorithms are often iterative



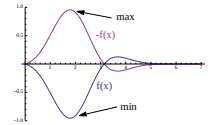
goes with audio02.mp3

Maximize vs Minimize



- from now on we'll talk about minimization

Maximize vs Minimize

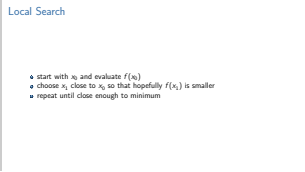


- from now on we'll talk about minimization

goes with audio03.mp3

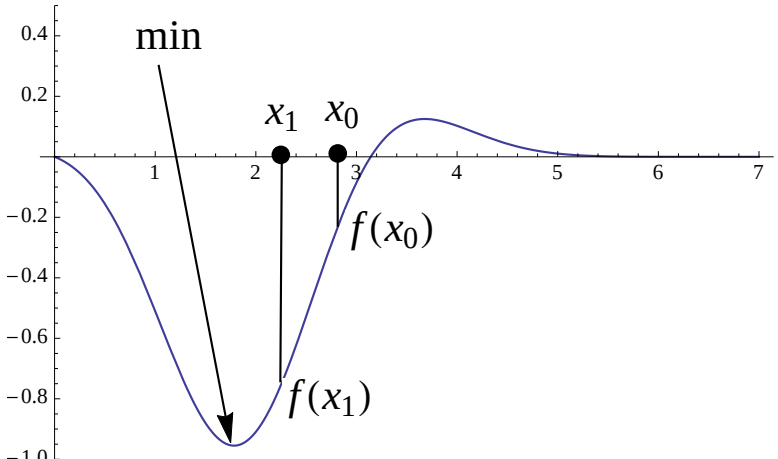
Local Search

- start with x_0 and evaluate $f(x_0)$
- choose x_1 close to x_0 so that hopefully $f(x_1)$ is smaller
- repeat until close enough to minimum

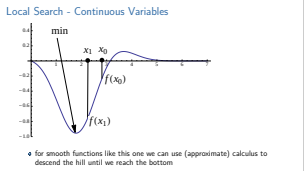


goes with audio04.mp3

Local Search - Continuous Variables

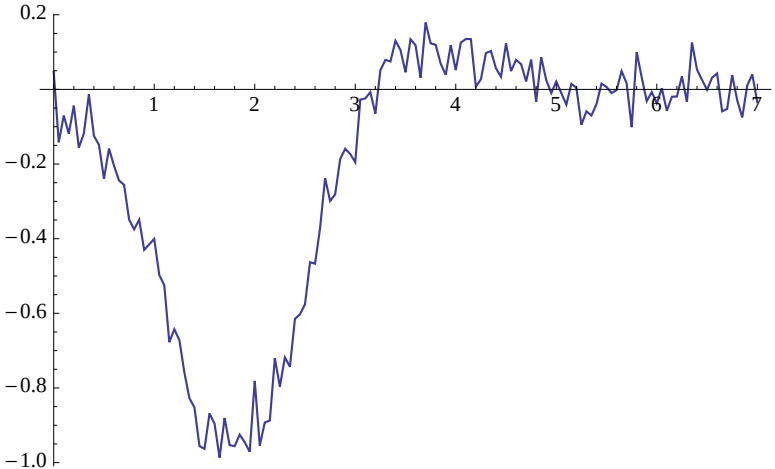


- for smooth functions like this one we can use (approximate) calculus to descend the hill until we reach the bottom

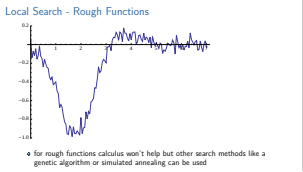


goes with audio05.mp3

Local Search - Rough Functions

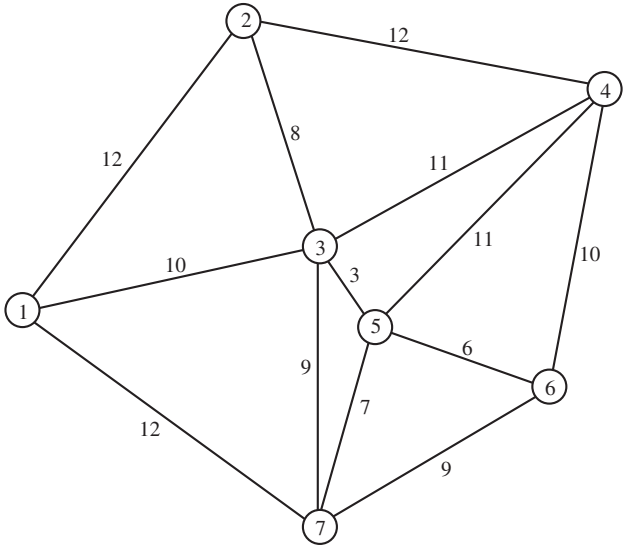


- for rough functions calculus won't help but other search methods like a genetic algorithm or simulated annealing can be used



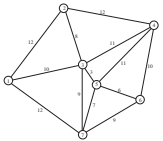
goes with audio06.mp3

Local Search - Discrete Variables



- Let $x_0 = 1\text{-}2\text{-}3\text{-}4\text{-}5\text{-}6\text{-}7\text{-}1$, with cost 69
- Swap cities 2 and 3 to get $x_1 = 1\text{-}3\text{-}2\text{-}4\text{-}5\text{-}6\text{-}7\text{-}1$, with cost 68
- Continue making swaps that improve the cost
- Stop when no improving swaps are possible
- No guarantee that result is best possible tour

Local Search - Discrete Variables

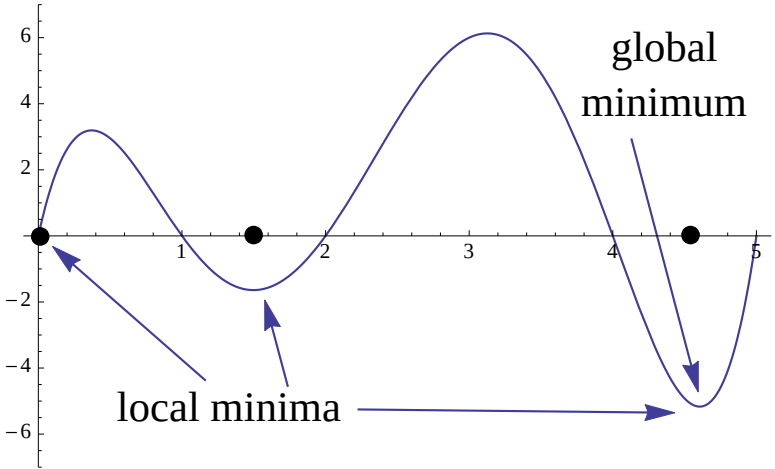


- Let $x_0 = 1\text{-}2\text{-}3\text{-}4\text{-}5\text{-}6\text{-}7\text{-}1$, with cost 69
- Swap cities 2 and 3 to get $x_1 = 1\text{-}3\text{-}2\text{-}4\text{-}5\text{-}6\text{-}7\text{-}1$, with cost 66
- Continue making swaps that improve the cost
- Stop when no improving swaps are possible
- No guarantee that result is best possible tour

goes with audio07.mp3

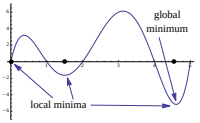
Local or Global Minimum?

Minimize $f(x) = 0.5x^5 - 6x^4 + 24.5x^3 - 39x^2 + 20x$ subject to $0 \leq x \leq 5$.



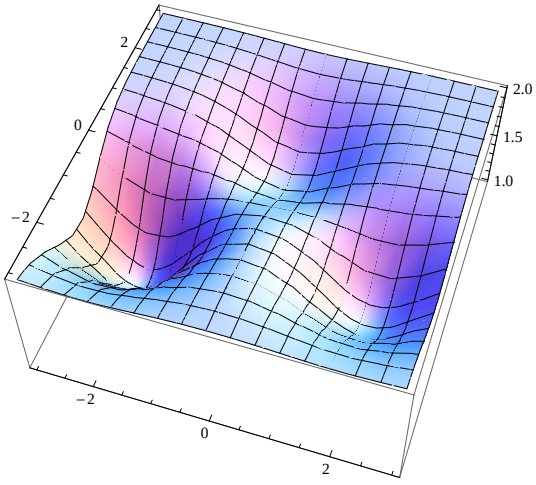
Local or Global Minimum?

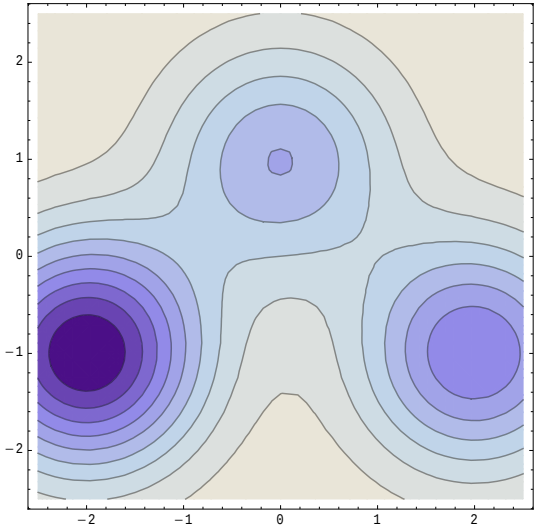
Minimize $f(x) = 0.5x^5 - 6x^4 + 24.5x^3 - 39x^2 + 20x$ subject to $0 \leq x \leq 5$.



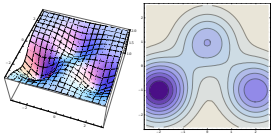
goes with audio08.mp3

Local vs Global again





Local vs Global again



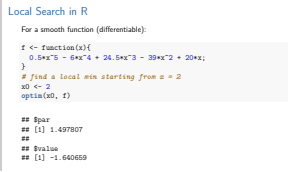
goes with audio09.mp3

Local Search in R

For a smooth function (differentiable):

```
f <- function(x){
  0.5*x^5 - 6*x^4 + 24.5*x^3 - 39*x^2 + 20*x;
}
# find a local min starting from x = 2
x0 <- 2
optim(x0, f)
```

```
## $par
## [1] 1.497807
##
## $value
## [1] -1.640659
```


Local Search in R

For a smooth function (differentiable):

```
f <- function(x){
  0.5*x^5 - 6*x^4 + 24.5*x^3 - 39*x^2 + 20*x;
}
# find a local min starting from x = 2
x0 <- 2
optim(x0, f)

## $par
## [1] 1.497807
##
## $value
## [1] -1.640659
```

- goes with audio10.mp3
- For a one-dimensional function, that is, one with a single continuous input variable, using optim() in R actually gives a warning to use optimize instead

optimize() for 1D function in R

```
f <- function(x){
  0.5*x^5 - 6*x^4 + 24.5*x^3 - 39*x^2 + 20*x;
}
optimize(f,c(0,5)) # specify the interval
```

```
## $minimum
## [1] 1.497809
##
## $objective
## [1] -1.640659
```

optimize() is supposed to find *the* minimum of a continuous f on $[a, b]$, but it's really just another local search. It will find other minima if you choose a good interval.

optimize() for 1D function in R

```
f <- function(x){
  0.5*x^5 - 6*x^4 + 24.5*x^3 - 39*x^2 + 20*x;
}
optimize(f,c(0,5)) # specify the interval
```

```
## $minimum
## [1] 1.497809
##
## $objective
## [1] -1.640659
```

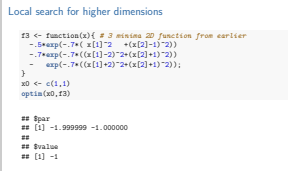
optimize() is supposed to find *the* minimum of a continuous f on $[a, b]$, but it's really just another local search. It will find other minima if you choose a good interval.

goes with audio11.mp3

Local search for higher dimensions

```
f3 <- function(x){ # 3 minima 2D function from earlier
  -.5*exp(-.7*( x[1]^2   +(x[2]-1)^2))
  -.7*exp(-.7*((x[1]-2)^2+(x[2]+1)^2))
  -   exp(-.7*((x[1]+2)^2+(x[2]+1)^2));
}
x0 <- c(1,1)
optim(x0,f3)
```

```
## $par
## [1] -1.999999 -1.000000
##
## $value
## [1] -1
```



goes with audio12.mp3

optim() in package ‘stats’

- optim() is really a gradient based search method
- if the gradient is not supplied, it is approximated numerically
- for a smooth function for which the gradient is not available, it is probably better to use a “derivative-free” method from package ‘nloptr’
  - use newuoa() in ‘nloptr’ for unconstrained optimization
  - use bobyqa() in ‘nloptr’ for box constrained problems
  - use cobyla() for more complex constraints
- for large numbers of variables it really pays to get the gradient if possible, but the derivative-free algorithms can handle 100’s of variables

optim() in package 'stats'

- optim() is really a gradient based search method
- if the gradient is not supplied, it is approximated numerically
- for a smooth function for which the gradient is not available, it is probably better to use a "derivative-free" method from package 'nloptr'
  - use newuoa() in 'nloptr' for unconstrained optimization
  - use bobyqa() in 'nloptr' for box constrained problems
  - use cobyla() for more complex constraints
- for large numbers of variables it really pays to get the gradient if possible, but the derivative-free algorithms can handle 100's of variables

goes with audio13.mp3

nloptr package

- provides interface to open-source NLopt
- uniform syntax for many optimization algorithms
- can be a bit tricky to install (not required for homework)
  - some instructions in download packet (but Google is your friend)
  - for Windows get precompiled NLopt binaries and also Rtools
  - for Mac/Linux download NLopt source and compile
  - install.packages('nloptr') in R

nloptr package

- provides interface to open-source NLopt
- uniform syntax for many optimization algorithms
- can be a bit tricky to install (not required for homework)
  - some instructions in download packet (but Google is your friend)
  - for Windows get precompiled NLopt binaries and also Rtools
  - for Mac/Linux download NLopt source and compile
  - install.packages('nloptr') in R

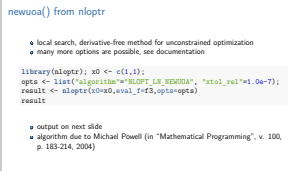
goes with audio14.mp3

newuoa() from nloptr

- local search, derivative-free method for unconstrained optimization
- many more options are possible, see documentation

```
library(nloptr); x0 <- c(1,1);
opts <- list("algorithm"="NLOPT_LN_NEWUOA", "xtol_rel"=1.0e-7);
result <- nloptr(x0=x0,eval_f=f3,opts=opts)
result
```

- output on next slide
- algorithm due to Michael Powell (in “Mathematical Programming”, v. 100, p. 183-214, 2004)



put below slide: there is no audio for this or the next three slides. if you can install nloptr, then you may wish to try the code yourself. The source code for these is embedded in the Week_09_Storybook.Rmd file in the download packet

newuoa() output

- happens to find global optimum even though it doesn't start nearby

```
## Minimization using NLopt version 2.4.2
##
## NLopt solver status: 1 ( NLOPT_SUCCESS: Generic success return valu
##
## Number of Iterations....: 47
## Termination conditions:  xtol_rel: 1e-07
## Number of inequality constraints:  0
## Number of equality constraints:    0
## Optimal value of objective function:  -1
## Optimal value of controls: -2 -1
```

bobyqa() from nloptr

- local search, derivative-free method for box constrained optimization
- many more options are possible, see documentation

```
library(nloptr)
opts <- list("algorithm"="NLOPT_LN_BOBYQA", "xtol_rel"=1.0e-7,
             lb=c(-2.5,-2.5),ub =c(2.5,2.5));
result <- nloptr(x0=x0,eval_f=f3,opts=opts)
result
```

- output on next slide
- algorithm due to Michael Powell (in "Mathematical Programming", v. 100, p. 183-214, 2004)

bobyqa() output

- happens to find global optimum even though it doesn't start nearby

```
## Minimization using NLopt version 2.4.2
##
## NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization stopped
## xtol_rel or xtol_abs (above) was reached. )
##
## Number of Iterations....: 44
## Termination conditions:  xtol_rel: 1e-07
## Number of inequality constraints:  0
## Number of equality constraints:    0
## Optimal value of objective function:  -1
## Optimal value of controls: -2 -1
```

Multistart Methods

Multistart Methods

- video01.mp4 in video folder This slide is a placeholder for a movie that illustrates trying different starting points and exploring the resulting local minima, then accepting the smallest one as our optimal solution. It may not be a global minimum.

Naive Multistart

- Choose a random sample of starting points in the feasible domain.
- Start a local search at each one.
- Return the coordinates and function value of the lowest minimum.

Naive Multistart

- Choose a random sample of starting points in the feasible domain.
- Start a local search at each one.
- Return the coordinates and function value of the lowest minimum.

- explore in HW

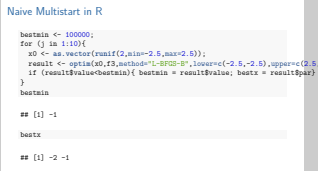
Naive Multistart in R

```
bestmin <- 100000;
for (j in 1:10){
  x0 <- as.vector(runif(2,min=-2.5,max=2.5));
  result <- optim(x0,f3,method="L-BFGS-B",lower=c(-2.5,-2.5),upper=c(
  if (result$value<bestmin){ bestmin = result$value; bestx = result$p
}
bestmin
```

```
## [1] -1
```

```
bestx
```

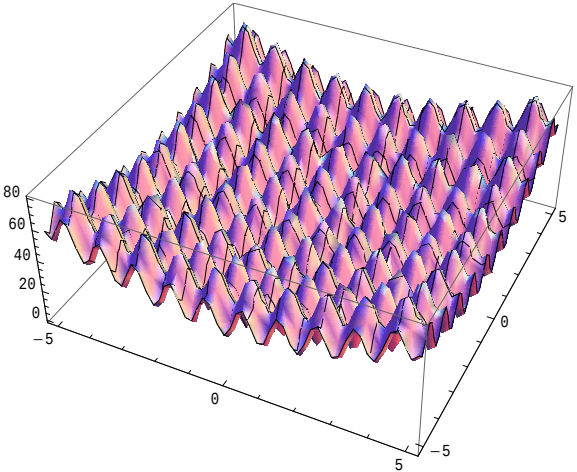
```
## [1] -2 -1
```



goes with audio15.mp3

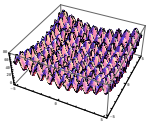
Rastrigin Function

in 2D $f(x, y) = 20 + x^2 + y^2 - 10\cos(2\pi x) - 10\cos(2\pi y), -5.12 \leq x, y \leq 5.12$



Rastrigin Function

in 2D $f(x, y) = 20 + x^2 + y^2 - 10\cos(2\pi x) - 10\cos(2\pi y), -5.12 \leq x, y \leq 5.12$



goes with audio16.mp3

Naive Multistart on Rastrigin

```
fr <- function(x){
  20 + x[1]^2 + x[2]^2 - 10*cos(2*pi*x[1]) - 10*cos(2*pi*x[2])
}
bestmin <- 100000; set.seed(126)
for (j in 1:50){
  x0 <- as.vector(runif(2,min=-5.12,max=5.12));
  result <- optim(x0,fr,method="L-BFGS-B",lower=c(-5.12,-5.12),upper=
  if (result$value<bestmin){ bestmin = result$value; bestx = result$p
}
bestmin
bestx
```

- output on next slide

Naive Multistart on Rastrigin

```
fr <- function(x){
  20 + x[1]^2 + x[2]^2 - 10*cos(2*pi*x[1]) - 10*cos(2*pi*x[2])
}
bestmin <- 100000; set.seed(126)
for (j in 1:50){
  x0 <- as.vector(runif(2,min=-5.12,max=5.12));
  result <- optim(x0,fr,method="L-BFGS-B",lower=c(-5.12,-5.12),upper=c(5.12,5.12))
  if (result$value<bestmin){ bestmin = result$value; bestx = result$par}
}
bestmin
bestx
```

- output on next slide

goes with audio17.mp3

Naive Multistart on Rastrigin - output

```
round(bestmin)
```

```
## [1] 1
```

```
round(bestx,4)
```

```
## [1] 0.000 0.995
```

Naive Multistart on Rastrigin - output

```
round(bestmin)
```

```
## [1] 1
```

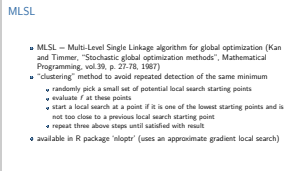
```
round(bestx,4)
```

```
## [1] 0.000 0.995
```

goes with audio18.mp3

MLSL

- MLSL = Multi-Level Single Linkage algorithm for global optimization (Kan and Timmer, "Stochastic global optimization methods", Mathematical Programming, vol.39, p. 27-78, 1987)
- "clustering" method to avoid repeated detection of the same minimum
  - randomly pick a small set of potential local search starting points
  - evaluate f at these points
  - start a local search at a point if it is one of the lowest starting points and is not too close to a previous local search starting point
  - repeat three above steps until satisfied with result
- available in R package 'nloptr' (uses an approximate gradient local search)



goes with audio19.mp3

MLSL applied to Rastigrin Function

```
require(nloptr); x0<-c(5,5);
result <- mlsl(x0,fr,lower=c(-5.12,-5.12),upper=c(5.12,5.12),nl.info=
```

```
## NLopt solver status: 5 ( NLOPT_MAXEVAL_REACHED: Optimization stopp
## because maxeval (above) was reached. )
##
## Number of Iterations....: 1001
## Termination conditions:  xtol_rel: 1e-06 maxeval: 1000   ftol_rel:
## Number of inequality constraints:  0
## Number of equality constraints:    0
## Current value of objective function:  0
## Current value of controls: 3.529593e-13 -3.529593e-13
```

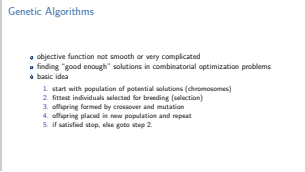
MLSL applied to Rastigrin Function

```
require(nloptr); x0<-c(5,5);
result <- mlsl(x0,fr,lower=c(-5.12,-5.12),upper=c(5.12,5.12),ml.info=TRUE)

## NLopt solver status: 5 ( NLOPT_MAXEVAL_REACHED: Optimization stopped
## because maxeval (above) was reached. )
##
## Number of Iterations....: 1001
## Termination conditions:  xtol_rel: 1e-06 maxeval: 1000   ftol_rel: 0 ftol_abs: 0
## Number of inequality constraints:  0
## Number of equality constraints:    0
## Current value of objective function:  0
## Current value of controls: 3.529593e-13 -3.529593e-13
```

goes with audio20.mp3

Genetic Algorithms

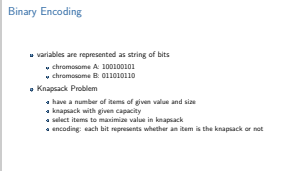
- objective function not smooth or very complicated
- finding "good enough" solutions in combinatorial optimization problems
- basic idea
  1. start with population of potential solutions (chromosomes)
  2. fittest individuals selected for breeding (selection)
  3. offspring formed by crossover and mutation
  4. offspring placed in new population and repeat
  5. if satisfied stop, else goto step 2.



- goes with audio21.mp3 PUT BELOW SLIDE: in addition to reading the textbook, there is a very good introduction at `http://www.obitko.com/tutorials/genetic-algorithms/index.php`

Binary Encoding

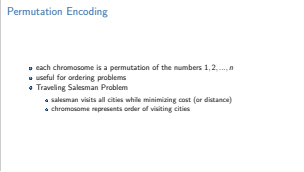
- variables are represented as string of bits
  - chromosome A: 100100101
  - chromosome B: 011010110
- Knapsack Problem
  - have a number of items of given value and size
  - knapsack with given capacity
  - select items to maximize value in knapsack
  - encoding: each bit represents whether an item is the knapsack or not



goes with audio22.mp3

Permutation Encoding

- each chromosome is a permutation of the numbers $1, 2, ..., n$
- useful for ordering problems
- Traveling Salesman Problem
  - salesman visits all cities while minimizing cost (or distance)
  - chromosome represents order of visiting cities



goes with audio23.mp3

Value Encoding

- each chromosome is string of some values such as reals, integers, characters
- real chromosomes can be used for complex optimization problems such as Rastrigin
- in the homework is an integer chromosome problem where integers represent assignment of cities to numbered congressional districts

Value Encoding

- each chromosome is string of some values such as reals, integers, characters
- real chromosomes can be used for complex optimization problems such as Rastrigin
- in the homework is an integer chromosome problem where integers represent assignment of cities to numbered congressional districts

goes with audio24.mp3

Genetic Algorithms in R

- the 'GA' package is very good and can deal with binary, permutation, and real encoding
- the 'gramEvol' package has a useful genetic algorithm for integer encoded problems
- both can be installed easily using `install.packages('name of package')`

Genetic Algorithms in R

- the 'GA' package is very good and can deal with binary, permutation, and real encoding
- the 'gramEvol' package has a useful genetic algorithm for integer encoded problems
- both can be installed easily using `install.packages('name of package')`

there is no audio for this one

GA for Knapsack Problem in R

```
value = c(4,6,5,3,6,3,7,8,9,10);
size = c(3,2,5,3,2,4,3,2,4,2); capacity = 15;
totalValue = function(xb){
  sum(value*xb)*(sum(size*xb)<=capacity)
}
require(GA)
result = ga("binary",fitness=totalValue,nBits=10) # approx *max*
result@fitnessValue # total value of best knapsack
result@solution # binary vector showing which items to include
sum(size*result@solution) # capacity used
```

- output on next slide
- run this a few times, you won't always get the same solution!

GA for Knapsack Problem in R

```
value = c(4,6,5,3,6,3,7,8,9,10);
size = c(3,2,5,3,2,4,3,2,4,2); capacity = 15;
totalValue = function(xb){
  sum(value*xb)*(sum(size*xb)<=capacity)
}
require(GA)
result = ga("binary",fitness=totalValue,nBits=10) # approx *max*
result@fitnessValue # total value of best knapsack
result@solution # binary vector showing which items to include
sum(size*result@solution) # capacity used
```

- output on next slide
- run this a few times, you won't always get the same solution!

goes with audio25.mp3

Knapsack output

```
## [1] 46
```

```
##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
## [1,]  0  1  0  0  1  0  1  1  1   1
```

```
## [1] 15
```

Metaheuristics and Optimization in R

Knapsack output

Knapsack output

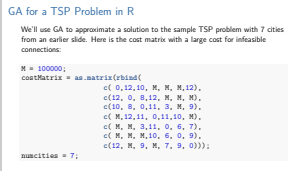
```
## [1] 46

##      x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
## [1,]  0  1  0  0  1  0  1  1  1   1

## [1] 15
```

no audio here

GA for a TSP Problem in R

We'll use GA to approximate a solution to the sample TSP problem with 7 cities from an earlier slide. Here is the cost matrix with a large cost for infeasible connections:

```
M = 100000;
costMatrix = as.matrix(rbind(
                    c( 0,12,10, M, M, M,12),
                    c(12, 0, 8,12, M, M, M),
                    c(10, 8, 0,11, 3, M, 9),
                    c( M,12,11, 0,11,10, M),
                    c( M, M, 3,11, 0, 6, 7),
                    c( M, M, M,10, 6, 0, 9),
                    c(12, M, 9, M, 7, 9, 0)));
numcities = 7;
```


GA for a TSP Problem in R

We'll use GA to approximate a solution to the sample TSP problem with 7 cities from an earlier slide. Here is the cost matrix with a large cost for infeasible connections:

```
M = 100000;
costMatrix = as.matrix(rbind(
                    c( 0,12,10, M, M, M,12),
                    c(12, 0, 8,12, M, M, M),
                    c(10, 8, 0,11, 3, M, 9),
                    c( M,12,11, 0,11,10, M),
                    c( M, M, 3,11, 0, 6, 7),
                    c( M, M, M,10, 6, 0, 9),
                    c(12, M, 9, M, 7, 9, 0)));
numcities = 7;
```

goes with audio26.mp3

GA for TSP

Here is the fitness function:

```
# given a tour, calculate the total cost
tourCost <- function(tour, costMatrix) {
  tour <- c(tour, tour[1])
  route <- embed(tour, 2)[, 2:1]
  sum(costMatrix[route])
}
# inverse of the total distance is the fitness
tspFitness <- function(tour, ...) 1/tourCost(tour, ...)
```

GA for TSP

Here is the fitness function:

```
# given a tour, calculate the total cost
tourCost <- function(tour, costMatrix) {
  tour <- c(tour, tour[1])
  route <- embed(tour, 2)[, 2:1]
  sum(costMatrix[route])
}
# inverse of the total distance is the fitness
tspFitness <- function(tour, ...) 1/tourCost(tour, ...)
```

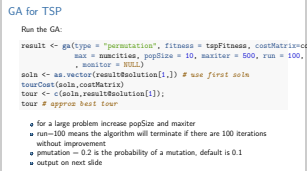
goes with audio27.mp3

GA for TSP

Run the GA:

```
result <- ga(type = "permutation", fitness = tspFitness, costMatrix=c
             max = numcities, popSize = 10, maxiter = 500, run = 100,
             , monitor = NULL)
soln <- as.vector(result@solution[1,]) # use first soln
tourCost(soln,costMatrix)
tour <- c(soln,result@solution[1]);
tour # approx best tour
```

- for a large problem increase popSize and maxiter
- run=100 means the algorithm will terminate if there are 100 iterations without improvement
- pmutation = 0.2 is the probability of a mutation, default is 0.1
- output on next slide


GA for TSP

Run the GA:

```
result <- ga(type = "permutation", fitness = tspFitness, costMatrix=costMatrix, min = 1,
             max = numcities, popSize = 10, maxiter = 500, run = 100, pmutation = 0.2
             , monitor = NULL)
soln <- as.vector(result@solution[1,]) # use first soln
tourCost(soln,costMatrix)
tour <- c(soln,result@solution[1]);
tour # approx best tour
```

- for a large problem increase popSize and maxiter
- run=100 means the algorithm will terminate if there are 100 iterations without improvement
- pmutation = 0.2 is the probability of a mutation, default is 0.1
- output on next slide

goes with audio28.mp3

GA for TSP

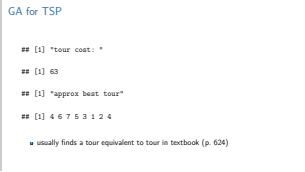
```
## [1] "tour cost: "
```

```
## [1] 63
```

```
## [1] "approx best tour"
```

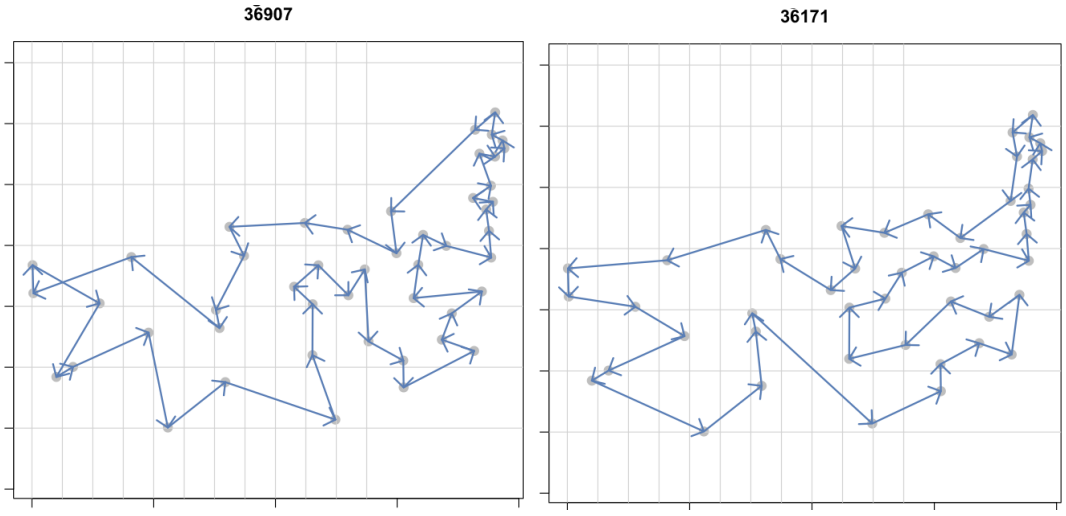
```
## [1] 4 6 7 5 3 1 2 4
```

- usually finds a tour equivalent to tour in textbook (p. 624)

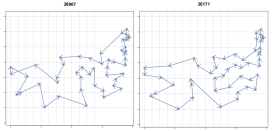


goes with audio29.mp3

GA for a larger TSP in R



GA for a larger TSP in R



goes with audio30.mp3

GA for the Rastrigin Problem

```
Rastrigin <- function(x) {
            -(sum(x^2 - 10 * cos(2 * pi  * x)) + 10 * length(x))
        }
dimension = 10;
lower = rep(-5.12,dimension);
upper = rep(5.12,dimension);
result = ga(type="real-valued",fitness=Rastrigin,min=lower,max=upper,
round(result@solution,5)
result@fitnessValue
```

- results on next slide

GA for the Rastrigin Problem

```
Rastrigin <- function(x) {
          -(sum(x^2 - 10 * cos(2 * pi  * x)) + 10 * length(x))
       }
dimension = 10;
lower = rep(-5.12,dimension);
upper = rep(5.12,dimension);
result = ga(type="real-valued",fitness=Rastrigin,min=lower,max=upper,maxiter=500)
round(result@solution,5)
result@fitnessValue
```

- results on next slide

goes with audio31.mp3

GA Rastrigin Results

- real-valued GA explores large solution space well, but converges slowly near optimum

```
##             x1       x2     x3       x4       x5       x6      x7     x
## [1,] -0.00619 0.00506 0.0036 0.00033 0.00053 -0.00273 0.00482 7e-04
##            x10
## [1,] -0.00053

## [1] -0.0227812
```

GA Rastrigin Results

- real-valued GA explores large solution space well, but converges slowly near optimum

```
##            x1       x2      x3       x4       x5          x6      x7      x8      x9
## [1,] -0.00619 0.00506 0.0036 0.00033 0.00053 -0.00273 0.00482 7e-04 0.0025
##           x10
## [1,] -0.00053

## [1] -0.0227812
```

goes with audio32.mp3

GA plus local search

- add optim=TRUE to ga() call to use optim() to do local search using each chromosome as an initial value to determine fitness

```
result = ga(type="real-valued",fitness=Rastrigin,
            min=lower,max=upper,maxiter=1000,optim=TRUE)
```

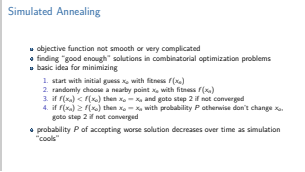
```
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
## [1,]  0  0  0  0  0  0  0  0  0   0
```

```
## [1] 0
```

goes with audio33.mp3

Simulated Annealing

- objective function not smooth or very complicated
- finding "good enough" solutions in combinatorial optimization problems
- basic idea for minimizing
  1. start with initial guess x_o with fitness $f(x_n)$
  2. randomly choose a nearby point x_n with fitness $f(x_n)$
  3. if $f(x_n) < f(x_o)$ then $x_o = x_n$ and goto step 2 if not converged
  4. if $f(x_n) \geq f(x_o)$ then $x_o = x_n$ with probability P otherwise don't change x_o, goto step 2 if not converged
- probability P of accepting worse solution decreases over time as simulation "cools"



goes with audio34.mp3

BELOW SLIDE: this video shows an animation of simulated annealing that I find really helpful in understanding how simulated annealing works. Be sure to read the text below the video for an explanation. (Include this link please)
`https://youtu.be/iaq_Fpr4KZc`

Simulated Annealing in R

- simulated annealing is an option in optim() in package 'stats'
- 'GenSA' package (check out the example of 30 dimensional Rastrigrin in documentation - hard problem!)

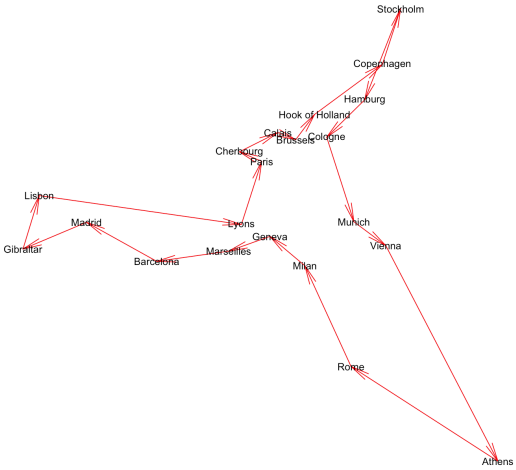
Simulated Annealing in R

- simulated annealing is an option in optim() in package 'stats'
- 'GenSA' package (check out the example of 30 dimensional Rastrigrin in documentation - hard problem!)

no audio here

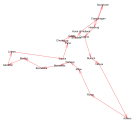
SA for TSP Problem in R

- see saTSP.R in download packet for the code



SA for TSP Problem in R

- see saTSP.R in download packet for the code



goes with audio35.mp3

SA for the Rastrigin Problem

```
Rastrigin <- function(x) {
            (sum(x^2 - 10 * cos(2 * pi  * x)) + 10 * length(x))
        }
dimension = 10;
lower = rep(-5.12,dimension); upper = rep(5.12,dimension);
x0 = runif(dimension,-5.12,5.12)
require(GenSA)
result = GenSA(x0,Rastrigin,lower=lower,upper=upper,
               control=list(max.call=5000))
result$value
result$par
```

- output on next slide

SA for the Rastrigin Problem

```
Rastrigin <- function(x) {
                (sum(x^2 - 10 * cos(2 * pi  * x)) + 10 * length(x))
        }
dimension = 10;
lower = rep(-5.12,dimension); upper = rep(5.12,dimension);
x0 = runif(dimension,-5.12,5.12)
require(GenSA)
result = GenSA(x0,Rastrigin,lower=lower,upper=upper,
               control=list(max.call=5000))
result$value
result$par
```

- output on next slide

goes with audio36.mp3

SA for Rastrigin output

```
## Loading required package: GenSA

## [1] "approximate minimum value"

## [1] 0.995

## [1] "occurs at"

##  [1] 0.000 0.000 0.000 0.995 0.000 0.000 0.000 0.000 0.000 0.000
```

SA for Rastrigin output

```
## Loading required package: GenSA
## [1] "approximate minimum value"
## [1] 0.995
## [1] "occurs at"
##  [1] 0.000 0.000 0.000 0.995 0.000 0.000 0.000 0.000 0.000 0.000
```

no audio