

Metaheuristics and Optimization in R

DS 775

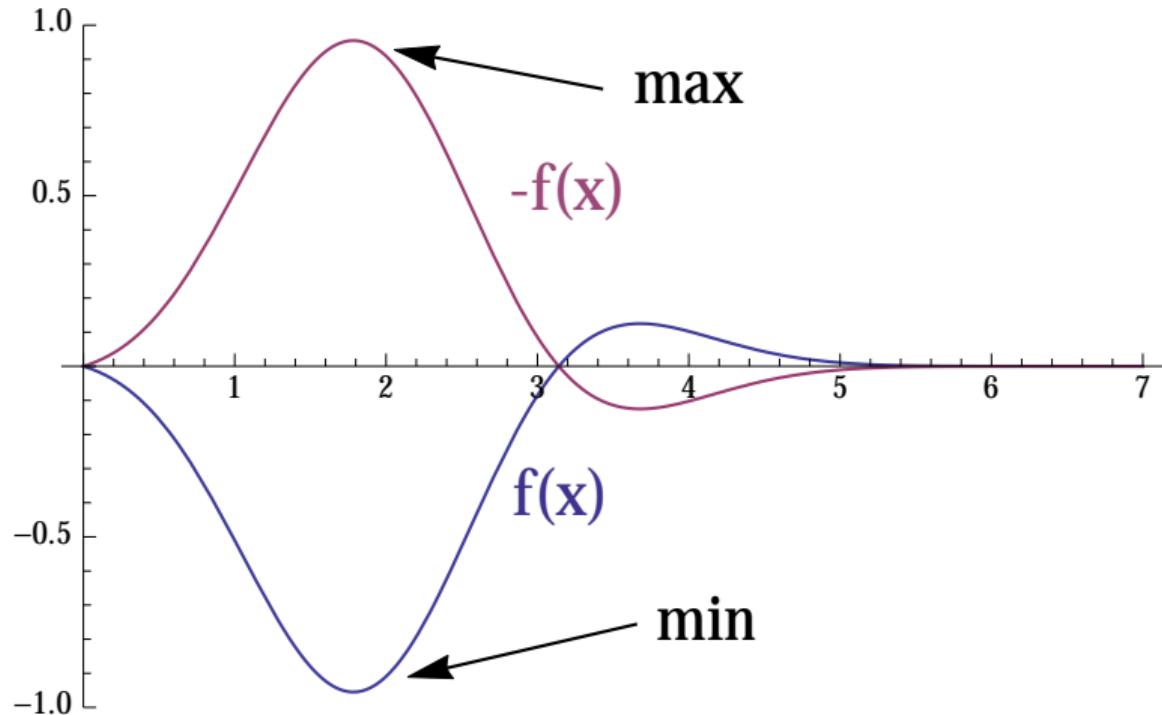
This Unit

- 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()`

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

Maximize vs Minimize

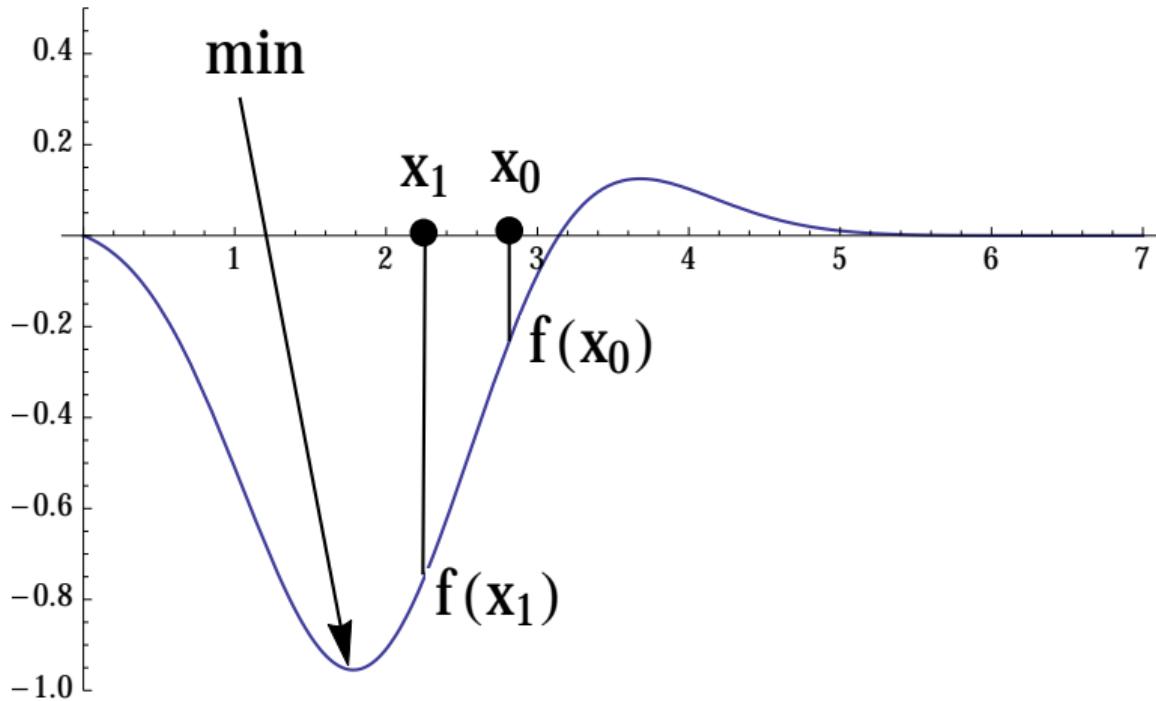


- from now on we'll talk about minimization

Local Search

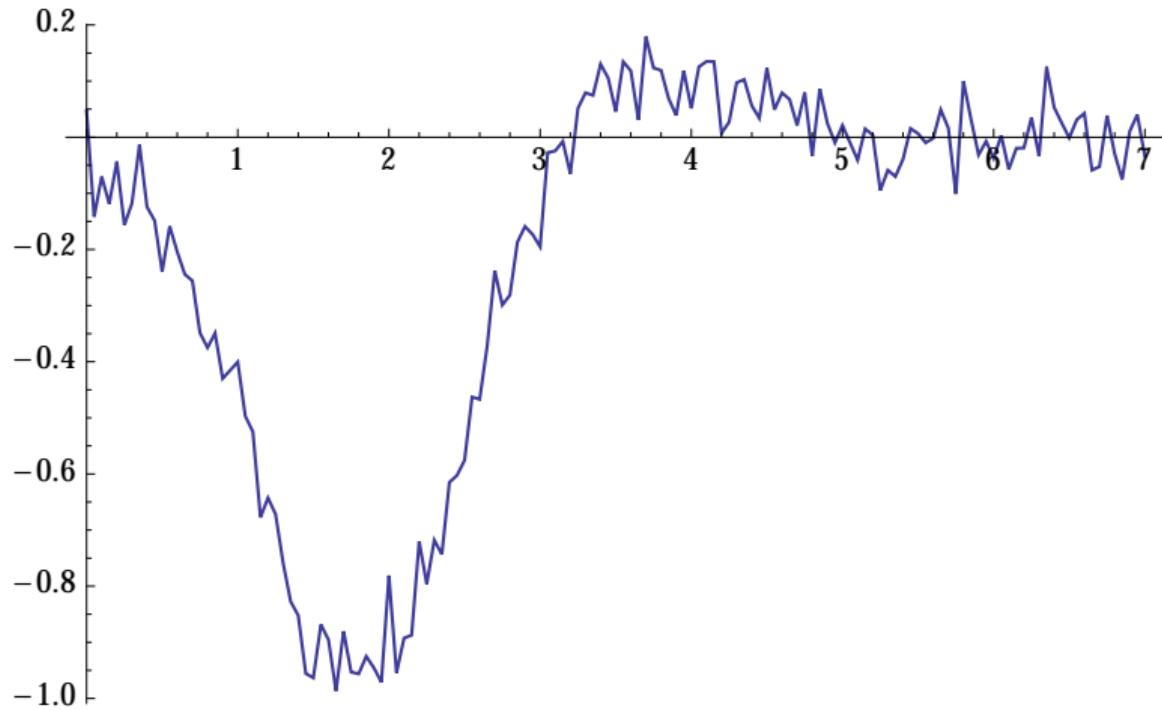
- start with x_0 and evaluate $f(x_0)$
- choose x_1 close to x_0 so that hopefully $f(x_1)$ is smaller
- repeat until close enough to minimum

Local Search - Continuous Variables



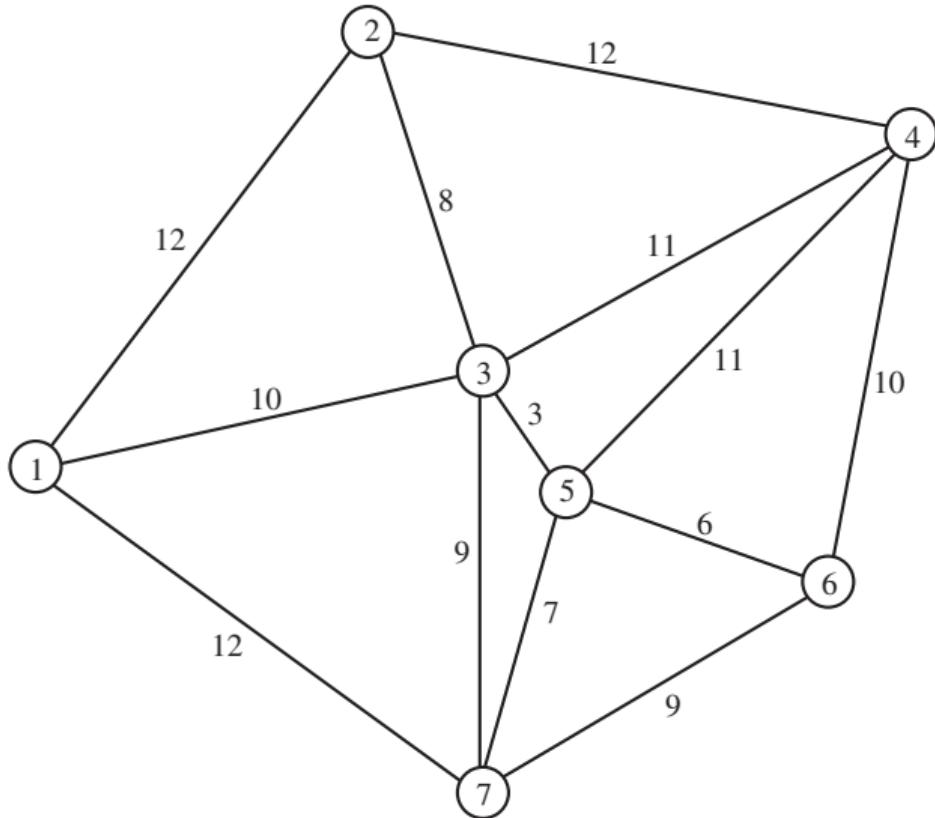
- for smooth functions like this one we can use (approximate) calculus to descend the hill until we reach the bottom

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

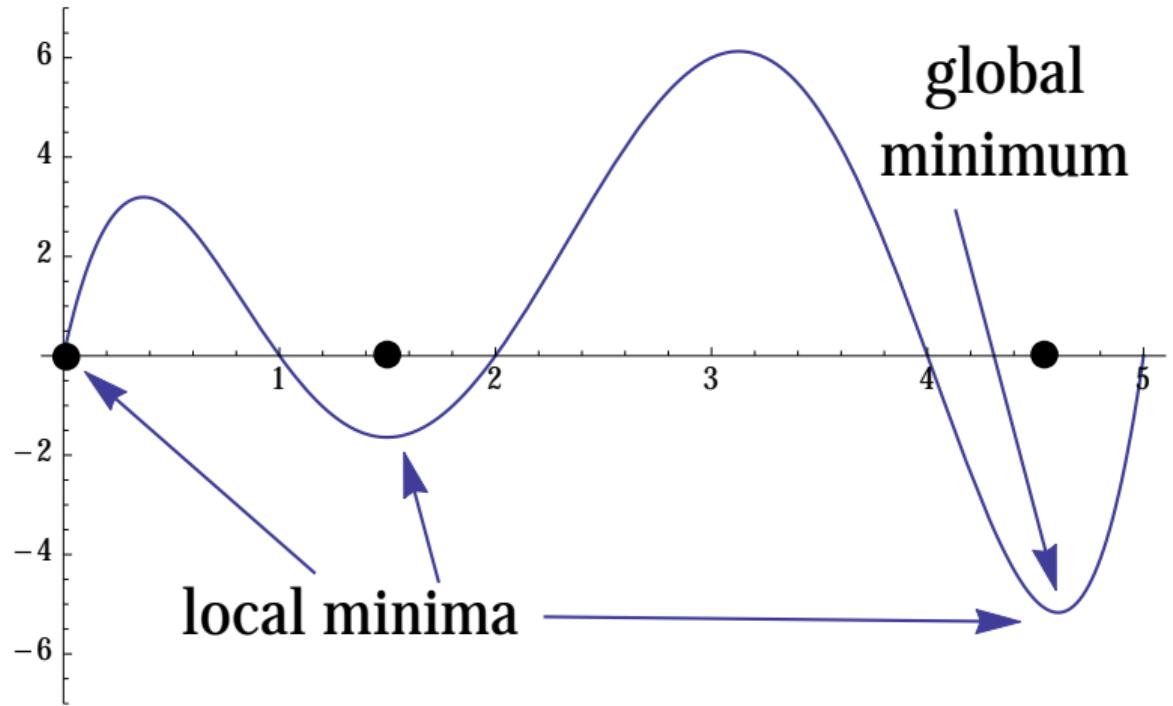
Local Search - Discrete Variables



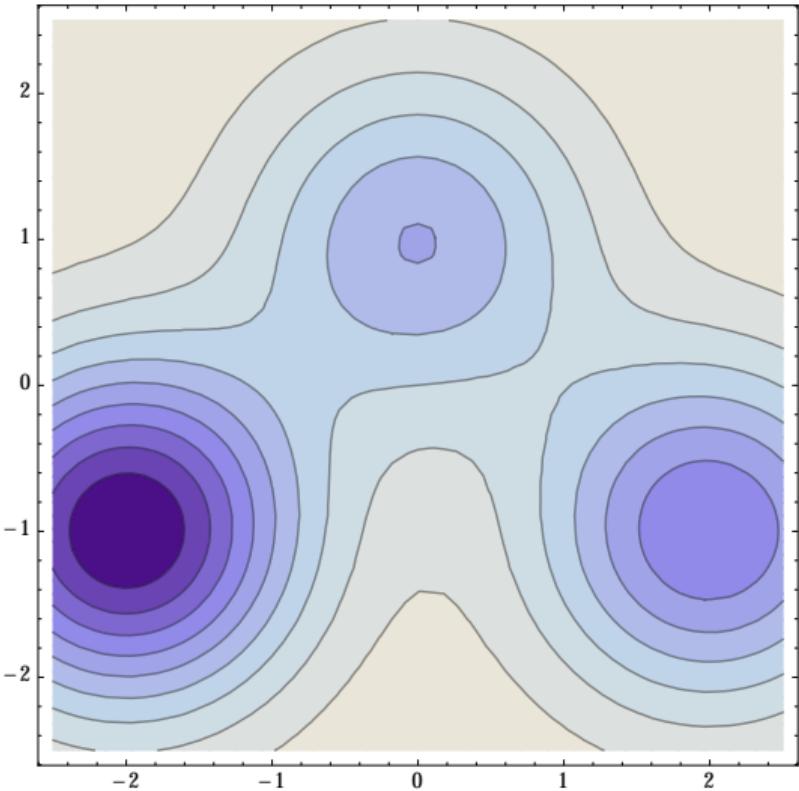
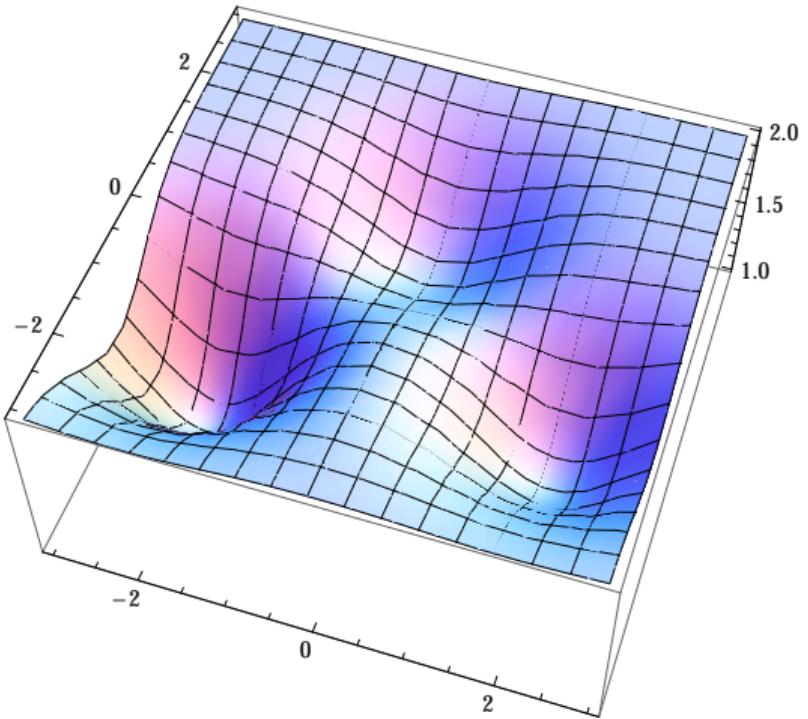
- Let $x_0 = 1-2-3-4-5-6-7-1$, with cost 69
- Swap cities 2 and 3 to get $x_1 = 1-3-2-4-5-6-7-1$, with cost 68
- Continue making swaps that improve the cost
- Stop when no improving swaps are possible
- No guarantee that result is best possible tour

Local 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 vs Global again



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
```

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.

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
```

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

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

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)

newuo() 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 value)
##
## 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

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 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$par
  }
bestmin

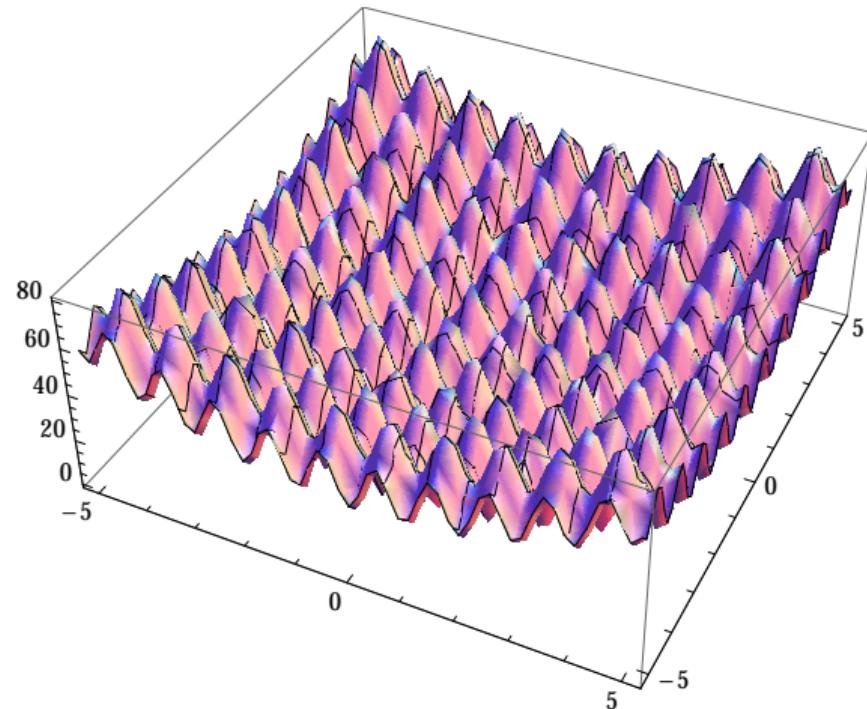
## [1] -1

bestx

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

Rastrigin Function

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



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 - output

```
round(bestmin)
```

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

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

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

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)

MLSL applied to Rastigrin Function

```
require(nloptr); x0<-c(5,5);
result <- mls1(x0,fr,lower=c(-5.12,-5.12),upper=c(5.12,5.12),nl.info=1)

## 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: 1e-06
## 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
```

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.

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 in the knapsack or not

Permutation Encoding

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

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

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')`

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!

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
```

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 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

Run the GA:

```
result <- ga(type = "permutation", fitness = tspFitness, costMatrix=costMatrix,
              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

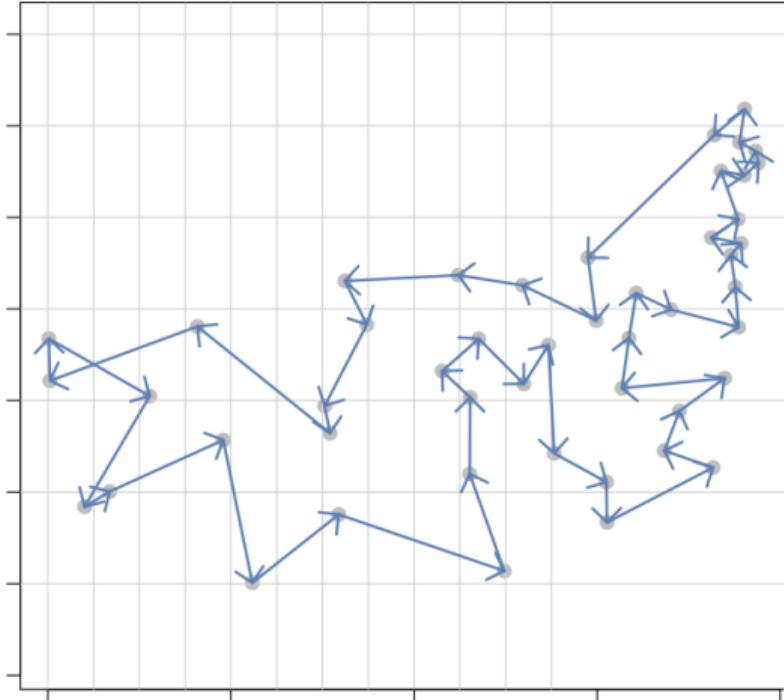
GA for TSP

```
## [1] "tour cost: "
## [1] 63
## [1] "approx best tour"
## [1] 4 6 7 5 3 1 2 4
```

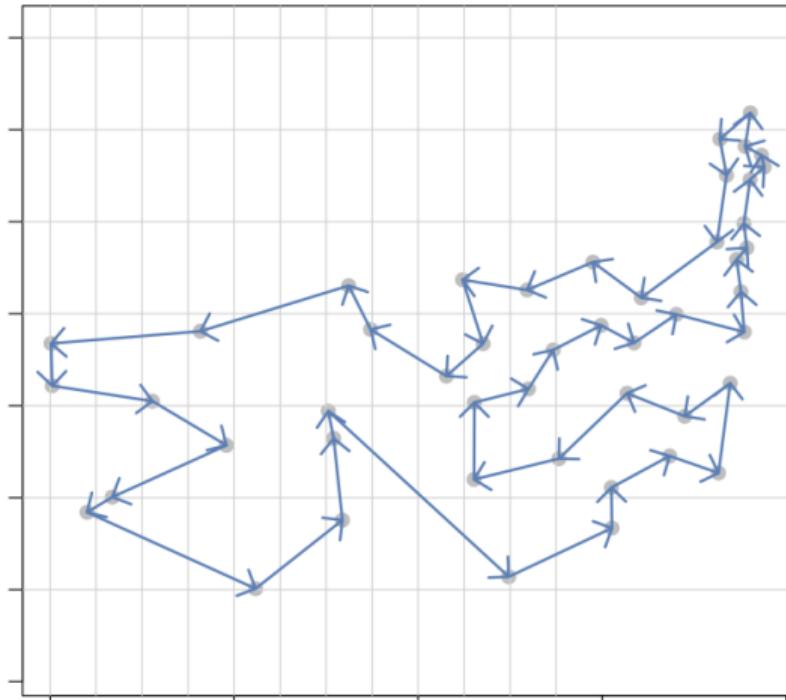
- usually finds a tour equivalent to tour in textbook (p. 624)

GA for a larger TSP in R

36907



36171



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 Rastrigin Results

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

```
##           x1          x2          x3          x4          x5          x6          x7          x8          x9          x10         x11         x12         x13         x14         x15         x16         x17         x18         x19         x20         x21         x22         x23         x24         x25         x26         x27         x28         x29         x30         x31         x32         x33         x34         x35         x36         x37         x38         x39         x40         x41         x42         x43         x44         x45         x46         x47         x48         x49         x50         x51         x52         x53         x54         x55         x56         x57         x58         x59         x60         x61         x62         x63         x64         x65         x66         x67         x68         x69         x70         x71         x72         x73         x74         x75         x76         x77         x78         x79         x80         x81         x82         x83         x84         x85         x86         x87         x88         x89         x90         x91         x92         x93         x94         x95         x96         x97         x98         x99         x100        x101        x102        x103        x104        x105        x106        x107        x108        x109        x110        x111        x112        x113        x114        x115        x116        x117        x118        x119        x120        x121        x122        x123        x124        x125        x126        x127        x128        x129        x130        x131        x132        x133        x134        x135        x136        x137        x138        x139        x140        x141        x142        x143        x144        x145        x146        x147        x148        x149        x150        x151        x152        x153        x154        x155        x156        x157        x158        x159        x160        x161        x162        x163        x164        x165        x166        x167        x168        x169        x170        x171        x172        x173        x174        x175        x176        x177        x178        x179        x180        x181        x182        x183        x184        x185        x186        x187        x188        x189        x190        x191        x192        x193        x194        x195        x196        x197        x198        x199        x200        x201        x202        x203        x204        x205        x206        x207        x208        x209        x210        x211        x212        x213        x214        x215        x216        x217        x218        x219        x220        x221        x222        x223        x224        x225        x226        x227        x228        x229        x230        x231        x232        x233        x234        x235        x236        x237        x238        x239        x240        x241        x242        x243        x244        x245        x246        x247        x248        x249        x250        x251        x252        x253        x254        x255        x256        x257        x258        x259        x260        x261        x262        x263        x264        x265        x266        x267        x268        x269        x270        x271        x272        x273        x274        x275        x276        x277        x278        x279        x280        x281        x282        x283        x284        x285        x286        x287        x288        x289        x290        x291        x292        x293        x294        x295        x296        x297        x298        x299        x299        x300        x301        x302        x303        x304        x305        x306        x307        x308        x309        x310        x311        x312        x313        x314        x315        x316        x317        x318        x319        x320        x321        x322        x323        x324        x325        x326        x327        x328        x329        x330        x331        x332        x333        x334        x335        x336        x337        x338        x339        x340        x341        x342        x343        x344        x345        x346        x347        x348        x349        x350        x351        x352        x353        x354        x355        x356        x357        x358        x359        x360        x361        x362        x363        x364        x365        x366        x367        x368        x369        x370        x371        x372        x373        x374        x375        x376        x377        x378        x379        x380        x381        x382        x383        x384        x385        x386        x387        x388        x389        x390        x391        x392        x393        x394        x395        x396        x397        x398        x399        x399        x400        x401        x402        x403        x404        x405        x406        x407        x408        x409        x410        x411        x412        x413        x414        x415        x416        x417        x418        x419        x420        x421        x422        x423        x424        x425        x426        x427        x428        x429        x430        x431        x432        x433        x434        x435        x436        x437        x438        x439        x440        x441        x442        x443        x444        x445        x446        x447        x448        x449        x450        x451        x452        x453        x454        x455        x456        x457        x458        x459        x460        x461        x462        x463        x464        x465        x466        x467        x468        x469        x470        x471        x472        x473        x474        x475        x476        x477        x478        x479        x480        x481        x482        x483        x484        x485        x486        x487        x488        x489        x490        x491        x492        x493        x494        x495        x496        x497        x498        x499        x499        x500        x501        x502        x503        x504        x505        x506        x507        x508        x509        x510        x511        x512        x513        x514        x515        x516        x517        x518        x519        x520        x521        x522        x523        x524        x525        x526        x527        x528        x529        x530        x531        x532        x533        x534        x535        x536        x537        x538        x539        x540        x541        x542        x543        x544        x545        x546        x547        x548        x549        x550        x551        x552        x553        x554        x555        x556        x557        x558        x559        x560        x561        x562        x563        x564        x565        x566        x567        x568        x569        x570        x571        x572        x573        x574        x575        x576        x577        x578        x579        x580        x581        x582        x583        x584        x585        x586        x587        x588        x589        x589        x590        x591        x592        x593        x594        x595        x596        x597        x598        x599        x599        x600        x601        x602        x603        x604        x605        x606        x607        x608        x609        x610        x611        x612        x613        x614        x615        x616        x617        x618        x619        x620        x621        x622        x623        x624        x625        x626        x627        x628        x629        x630        x631        x632        x633        x634        x635        x636        x637        x638        x639        x640        x641        x642        x643        x644        x645        x646        x647        x648        x649        x650        x651        x652        x653        x654        x655        x656        x657        x658        x659        x660        x661        x662        x663        x664        x665        x666        x667        x668        x669        x669        x670        x671        x672        x673        x674        x675        x676        x677        x678        x679        x680        x681        x682        x683        x684        x685        x686        x687        x688        x689        x689        x690        x691        x692        x693        x694        x695        x696        x697        x698        x699        x699        x700        x701        x702        x703        x704        x705        x706        x707        x708        x709        x710        x711        x712        x713        x714        x715        x716        x717        x718        x719        x719        x720        x721        x722        x723        x724        x725        x726        x727        x728        x729        x729        x730        x731        x732        x733        x734        x735        x736        x737        x738        x739        x739        x740        x741        x742        x743        x744        x745        x746        x747        x748        x749        x749        x750        x751        x752        x753        x754        x755        x756        x757        x758        x759        x760        x761        x762        x763        x764        x765        x766        x767        x768        x769        x769        x770        x771        x772        x773        x774        x775        x776        x777        x778        x779        x779        x780        x781        x782        x783        x784        x785        x786        x787        x788        x789        x789        x790        x791        x792        x793        x794        x795        x796        x797        x798        x799        x799        x800        x801        x802        x803        x804        x805        x806        x807        x808        x809        x810        x811        x812        x813        x814        x815        x816        x817        x818        x819        x819        x820        x821        x822        x823        x824        x825        x826        x827        x828        x829        x829        x830        x831        x832        x833        x834        x835        x836        x837        x838        x839        x839        x840        x841        x842        x843        x844        x845        x846        x847        x848        x849        x849        x850        x851        x852        x853        x854        x855        x856        x857        x858        x859        x860        x861        x862        x863        x864        x865        x866        x867        x868        x869        x869        x870        x871        x872        x873        x874        x875        x876        x877        x878        x879        x879        x880        x881        x882        x883        x884        x885        x886        x887        x888        x889        x889        x890        x891        x892        x893        x894        x895        x896        x897        x898        x899        x899        x900        x901        x902        x903        x904        x905        x906        x907        x908        x909        x910        x911        x912        x913        x914        x915        x916        x917        x918        x919        x919        x920        x921        x922        x923        x924        x925        x926        x927        x928        x929        x929        x930        x931        x932        x933        x934        x935        x936        x937        x938        x939        x939        x940        x941        x942        x943        x944        x945        x946        x947        x948        x949        x949        x950        x951        x952        x953        x954        x955        x956        x957        x958        x959        x960        x961        x962        x963        x964        x965        x966        x967        x968        x969        x969        x970        x971        x972        x973        x974        x975        x976        x977        x978        x979        x979        x980        x981        x982        x983        x984        x985        x986        x987        x988        x989        x989        x990        x991        x992        x993        x994        x995        x996        x997        x998        x999        x999        x1000
```

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
```

Simulated Annealing

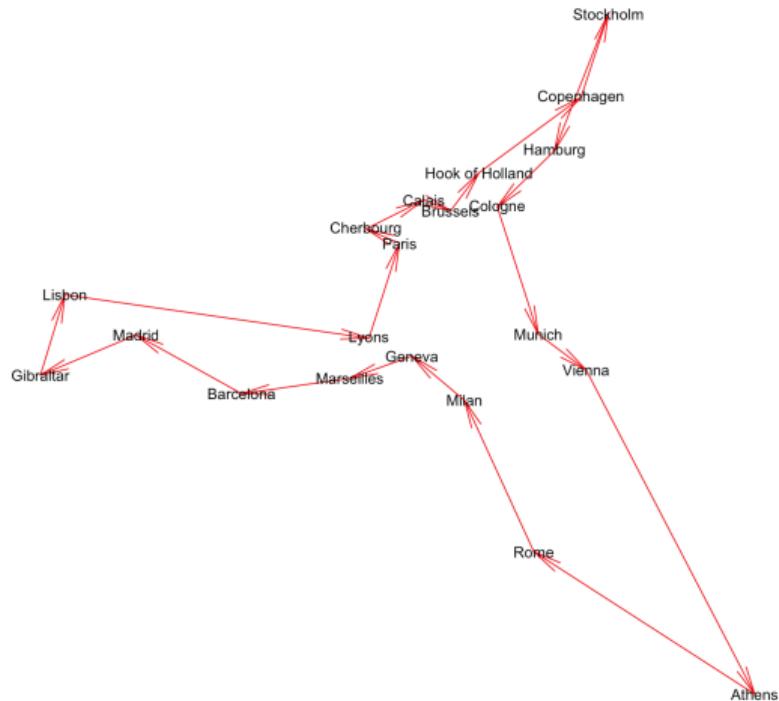
- objective function not smooth or very complicated
- finding “good enough” solutions in combinatorial optimization problems
- basic idea for minimizing
 1. start with initial guess x_o with fitness $f(x_n)$
 2. randomly choose a nearby point x_n with fitness $f(x_n)$
 3. if $f(x_n) < f(x_o)$ then $x_o = x_n$ 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”

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!)

SA for TSP Problem in R

- see saTSP.R in download packet for the code



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 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 0.000
```