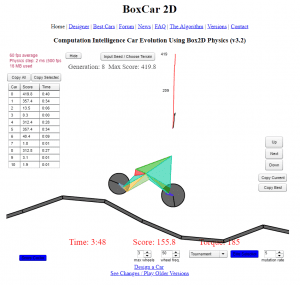
Source: Wikimedia

Hamlet: *Do you see yonder cloud that’s almost in shape of a camel?*  
Polonius: *By the mass, and ’tis like a camel, indeed.*  
Hamlet: *Methinks it is like a weasel.*  
*from* Hamlet *by* William Shakespeare

The best way to see how evolution works, is to watch it in action! You can watch the evolution of cars live in this application (but be careful, it’s addictive): [BoxCar 2D](http://boxcar2d.com/)

It is fascinating to see how those cars get better and better over time, sometimes finding very impressive solutions:



To understand how evolution works even better, let us create an artificial evolution in R!

The famous evolutionary biologist Richard Dawkins gave in his book “The Blind Watchmaker” the following thought experiment:

I don’t know who it was first pointed out that, given enough time, a monkey bashing away at random on a typewriter could produce all the works of Shakespeare. The operative phrase is, of course, given enough time. Let us limit the task facing our monkey somewhat. Suppose that he has to produce, not the complete works of Shakespeare but just the short sentence ‘Methinks it is like a weasel’, and we shall make it relatively easy by giving him a typewriter with a restricted keyboard, one with just the 26 (capital) letters, and a space bar. How long will he take to write this one little sentence?

We are now going to put this idea into practice! The following outline is from the Wikipedia article on the weasel program ([Weasel program](https://en.wikipedia.org/wiki/Weasel_program)):

1. Start with a random string of 28 characters.
2. Make 100 copies of the string (reproduce).
3. For each character in each of the 100 copies, with a probability of 5%, replace (mutate) the character with a new random character.
4. Compare each new string with the target string “METHINKS IT IS LIKE A WEASEL”, and give each a score (the number of letters in the string that are correct and in the correct position).
5. If any of the new strings has a perfect score (28), halt. Otherwise, take the highest scoring string, and go to step 2.

So let us first define some variables and helper functions for *reproduction*, *mutation* and *fitness* calculation:

target <- unlist(strsplit("METHINKS IT IS LIKE A WEASEL" , "")) # assign target string to "target"

pop\_sz <- 100 # assign population size 100 to "pop\_sz"

mt\_rt <- 0.05 # assign mutation rate 5% to "mt\_rt"

reproduce <- function(string) {

# input: vector "string"

# output: matrix with "pop\_sz" columns, where each column is vector "string"

matrix(string, nrow = length(string), ncol = pop\_sz)

}

mutate <- function(pop) {

# input: matrix of population "pop"

# output: matrix of population where each character, with a probability of mt\_rt per cent (= 5%), is replaced with a new random character

mt\_pos <- runif(length(pop)) <= mt\_rt

pop[mt\_pos] <- sample(c(LETTERS, " "), sum(mt\_pos), replace = TRUE)

pop

}

fitness <- function(pop) {

# input: matrix of population "pop"

# output: vector of the number of letters that are correct (= equal to target) for each column

colSums(pop == target)

}

After that we are going through all five steps listed above:

# 1. Start with a random string of 28 characters.

set.seed(70)

start <- sample(c(LETTERS, " "), length(target), replace = TRUE)

# 2. Make 100 copies of this string (reproduce).

pop <- reproduce(start)

# 3. For each character in each of the 100 copies, with a probability of 5%, replace (mutate) the character with a new random character.

pop <- mutate(pop)

# 4. Compare each new string with the target "METHINKS IT IS LIKE A WEASEL", and give each a score (the number of letters in the string that are correct and in the correct position).

score <- fitness(pop)

# 5. If any of the new strings has a perfect score (28), halt. Otherwise, take the highest scoring string, and go to step 2.

highscorer <- pop[ , which.max(score)] # assign string to "highscorer" which has max. score in the population

gen\_no <- 1 #assign 1 to generation counter "gen\_no"

while (max(score) < length(target)) {

cat("No. of generations: ", gen\_no, ", best so far: ", highscorer, " with score: ", max(score), "\n", sep = "")

pop <- reproduce(highscorer) # 2. select the highest scoring string for reproduction

pop <- mutate(pop) # 3. mutation

score <- fitness(pop) # 4. fitness calculation

highscorer <- pop[ , which.max(score)] # assign string to "highscorer" which has max. score in the population

gen\_no <- gen\_no + 1 # increment generation counter

}

## No. of generations: 1, best so far: BZRDXXINEIMYQVJWBFZKFCVUPFYL with score: 2

## No. of generations: 2, best so far: BZRDXNINEIMYQVJWBFZKFCVUPFYL with score: 3

## No. of generations: 3, best so far: BZRDXNINEIMYQVJWBFZKACVEPFYR with score: 4

## No. of generations: 4, best so far: BZRDININEIMYQBJWBFZKACVEPFYR with score: 5

## No. of generations: 5, best so far: BZRDININEIMYIBJWBFZKACVEPFYR with score: 6

## No. of generations: 6, best so far: BZRDININEIMYIBJLBFZKACVEPFYR with score: 7

## No. of generations: 7, best so far: BRRDININEIMYIBJLOFZKACVEPFYL with score: 8

## No. of generations: 8, best so far: BRRDININEIMYIZJLOFZKACVEAFYL with score: 9

## No. of generations: 9, best so far: BRRDINKNEIMYIZJLOFZKAT EAFYL with score: 10

## No. of generations: 10, best so far: BRRDINKNEIMYIZJLOFZKATVEASYL with score: 11

## No. of generations: 11, best so far: BRRDINKNEIMYIZJLOFEKATVEASYL with score: 12

## No. of generations: 12, best so far: BRRUINKNEIMYIZJLOFEKATVEASEL with score: 13

## No. of generations: 13, best so far: BERUINKNEIMYIZJLOFEKATVEASEL with score: 14

## No. of generations: 14, best so far: BERHINKNEIMYIZJLVFEKATVEASEL with score: 15

## No. of generations: 15, best so far: BERHINKNEIMQIZJLVFE ATVEASEL with score: 16

## No. of generations: 16, best so far: BERHINKNEIMQIZ LVFE ATVEASEL with score: 17

## No. of generations: 17, best so far: BERHINKNEIMQIZ LVFE ATVEASEL with score: 17

## No. of generations: 18, best so far: BERHINKNEIMQIZ LVFE ATVEASEL with score: 17

## No. of generations: 19, best so far: TERHINKNEIMQIZ LVFE ATDEASEL with score: 17

## No. of generations: 20, best so far: TERHINKNEIMQIZ LVFE ATDEASEL with score: 17

## No. of generations: 21, best so far: TERHINKNJISQIZ LVFE ATDEASEL with score: 17

## No. of generations: 22, best so far: TERHINKNJISQIZ LVFE A DEASEL with score: 18

## No. of generations: 23, best so far: TERHINKNJISQIZ LVFE A DEASEL with score: 18

## No. of generations: 24, best so far: TERHINKNJITQIZ LVFE A YEASEL with score: 19

## No. of generations: 25, best so far: TERHINKNJITQIZ LPFE A YEASEL with score: 19

## No. of generations: 26, best so far: TERHINKN ITQIZ LPFE A YEASEL with score: 20

## No. of generations: 27, best so far: MERHINKN ITQIZ LPFE A YEASEL with score: 21

## No. of generations: 28, best so far: MERHINKN IT IZ LPFE A YEASEL with score: 22

## No. of generations: 29, best so far: MERHINKN IT IS LPFE A YEASEL with score: 23

## No. of generations: 30, best so far: MERHINKN IT IS LPFE A YEASEL with score: 23

## No. of generations: 31, best so far: MERHINKN IT IS LPFE A YEASEL with score: 23

## No. of generations: 32, best so far: MERHINKN IT IS LAFE A WEASEL with score: 24

## No. of generations: 33, best so far: METHINKN IT IS LAFE A WEASEL with score: 25

## No. of generations: 34, best so far: METHINKN IT IS LAFE A WEASEL with score: 25

## No. of generations: 35, best so far: METHINKN IT IS LAFE A WEASEL with score: 25

## No. of generations: 36, best so far: METHINKN IT IS LAFE A WEASEL with score: 25

## No. of generations: 37, best so far: METHINKN IT IS LAFE A WEASEL with score: 25

## No. of generations: 38, best so far: METHINKU IT IS LIFE A WEASEL with score: 26

## No. of generations: 39, best so far: METHINKU IT IS LIFE A WEASEL with score: 26

## No. of generations: 40, best so far: METHINKU IT IS LIFE A WEASEL with score: 26

## No. of generations: 41, best so far: METHINKU IT IS LIKE A WEASEL with score: 27

## No. of generations: 42, best so far: METHINKU IT IS LIKE A WEASEL with score: 27

## No. of generations: 43, best so far: METHINKU IT IS LIKE A WEASEL with score: 27

## No. of generations: 44, best so far: METHINKU IT IS LIKE A WEASEL with score: 27

## No. of generations: 45, best so far: METHINKU IT IS LIKE A WEASEL with score: 27

cat("Mission accomplished in ", gen\_no, " generations: ", highscorer, sep = "")

## Mission accomplished in 46 generations: METHINKS IT IS LIKE A WEASEL

As you can see, the algorithm arrived at the target phrase pretty quickly. Now, you can try to tweak different parameter setting, like the population size or the mutation rate, and see what happens. You can of course also change the target phrase.

A minority of (often very religious) people reject the fact of evolution because they miss a crucial step: selection based on fitness. Selection gives evolution direction towards solutions that are better able to solve a certain problem. It is the exact opposite of pure randomness which many people still suspect behind evolution.

To see the difference the only thing we have to do is to comment out the line  
pop <- reproduce(highscorer) which selects the highest scoring string for reproduction. We can see that without selection there is no improvement to be seen and the algorithm would run “forever”:

## No. of generations: 1, best so far: UJGGZYOEDJMRADTQUXFWAVWPBGFX with score: 2

## No. of generations: 2, best so far: UHGGZQOEDJERAD QBXFSBRWPBGFX with score: 2

## No. of generations: 3, best so far: UNGDZYOEDSERADTQIXFSBVWPAGFX with score: 3

## No. of generations: 4, best so far: UHGGZQNEDJERAG QBXFSBRWPBGWX with score: 2

## No. of generations: 5, best so far: IDGGTJOELJERAETQBDFSBVWEBGFX with score: 2

## No. of generations: 6, best so far: IDGGTJOELJERNETQBDFSBVWEBGFX with score: 2

## No. of generations: 7, best so far: FNJGZYOESJERERTQGXGSBVWEBSFX with score: 3

## No. of generations: 8, best so far: UJGWZBOERJMUAQTQUXFVAVWKKSFX with score: 3

## No. of generations: 9, best so far: VETGRYOEYVVSAOTQBKOSTVPPGGFM with score: 3

## No. of generations: 10, best so far: VETGRYOEYVVSAOTQBKOSTVPPGGFM with score: 3

## No. of generations: 11, best so far: VETGRYOEYVVSAKTQBKOSTVPPGGFM with score: 3

## No. of generations: 12, best so far: IETGRYOTYVVDAKTQBKOCTVPPGGFM with score: 3

## No. of generations: 13, best so far: TTVVZOKDJERADELYXFKWGWXKGYO with score: 3

## No. of generations: 14, best so far: UNGWCYOZDEWRAD WKXKSBVWECGFX with score: 3

## No. of generations: 15, best so far: UNGWCYOZDEWRBD WKXKSBVWECGFX with score: 3

## No. of generations: 16, best so far: UNGSCYOZDEWRBD WKXKSAVCECGFX with score: 3

## No. of generations: 17, best so far: MXKGZYOMSJ RIOTQBLJSBVNPAGDL with score: 4

## No. of generations: 18, best so far: MXKGZYOMSJ RIOTQBLJSBVNPAGDL with score: 4

## No. of generations: 19, best so far: MXKGZYOMZJ RIOTQBLJSVVNPAGDL with score: 4

## No. of generations: 20, best so far: TTVVJGKDDERADELYJXKRGWEKGYU with score: 4

## No. of generations: 21, best so far: TTVVJGKDDERADELYDXBRGWEKGYU with score: 4

## No. of generations: 22, best so far: TTWVJGKDQERADELYDXBRGWEKGYU with score: 4

## No. of generations: 23, best so far: MXKGOYOMCJ RIOTQBLJIVVAPAJDG with score: 3

## No. of generations: 24, best so far: MXKGOYOMCJ RIOTQBLJIVVAPAJDG with score: 3

## No. of generations: 25, best so far: MXKGOYOMCJ RIOTQBLJIVVAPAJDG with score: 3

## No. of generations: 26, best so far: MXKGOYOMCJ RIOTQBLJIVVAPAJDG with score: 3

## No. of generations: 27, best so far: TNTUXYKJPJNDAITLAJTYBAWPMGGB with score: 4

## No. of generations: 28, best so far: MXKGOYOMCJ RIOTLBLJIVVAPAJDX with score: 4

## No. of generations: 29, best so far: MXKGOYOMCJ RIOTLBLJIVVAJAJDX with score: 4

## No. of generations: 30, best so far: TUTUYYKNPJNDAITLAJTYBAAPMOGB with score: 3

## No. of generations: 31, best so far: NGAFULYDZELWD QDPRSMPWYAPZH with score: 3

## No. of generations: 32, best so far: HKUOZSJSXDERS TLBHASAVGPBEJT with score: 3

## No. of generations: 33, best so far: NGAFULYDTELWD QDPRSMPWYAPZH with score: 3

## No. of generations: 34, best so far: HKUYMSJAXDERS TLBHA AVGPBEJT with score: 3

## No. of generations: 35, best so far: HKUYMSJAXDSRS TLBHA AVGPBEJT with score: 3

## No. of generations: 36, best so far: HKXYMSJYXDSRS TLBHA AVGPNEJT with score: 3

## No. of generations: 37, best so far: KNGABULYDTELWD QDORSFPWYAPZH with score: 3

## No. of generations: 38, best so far: LLCIZN EOISJ DHFIEGPXNWYMYOX with score: 4

## No. of generations: 39, best so far: LLCIZN EOISJ DHFIEXPXNWYMYOX with score: 4

## No. of generations: 40, best so far: MZN KMIESQRRILELIIILFIGRYRZZ with score: 4

## No. of generations: 41, best so far: ITQXZEKK SENLSCJXAKQ EKNCNUJ with score: 3

## No. of generations: 42, best so far: MELBV VEUBRKXSNHWGILBU JVLZX with score: 3

## No. of generations: 43, best so far: DZNAKMIEOQRRILELIVILKIGVYRZZ with score: 3

## No. of generations: 44, best so far: DZNAKMIEOQRRILELIVILKIGVYRZZ with score: 3

## No. of generations: 45, best so far: LRPDILXMGCWDD ZQD BKANWHMKFI with score: 3

## No. of generations: 46, best so far: KEGAMRLYDAELDDUXLORSFPWOAPLH with score: 3

## No. of generations: 47, best so far: KEGAMRLYDAELDDUXLORSFPWOAPLH with score: 3

## No. of generations: 48, best so far: KEGAMRLYDAELDZUXLORHFPWOAPLH with score: 3

## No. of generations: 49, best so far: KEGAMRLYDAEWDZUXLORHFPWOAPLH with score: 3

## No. of generations: 50, best so far: KEGAMRLYDAEWDZDXLORHFPWOAPLH with score: 3

If this was how evolution really worked it wouldn’t work at all.

Because evolution is a very powerful *optimization* method there are also real world applications of so called *genetic algorithms* (GA). In the following example we want to find the *global optimum* of the so called Rastrigin function. What makes this task especially difficult for this popular test problem is the large number of *local minima*, as can be seen when plotting the function:

library(GA)

## Loading required package: foreach

## Loading required package: iterators

## Package 'GA' version 3.2

## Type 'citation("GA")' for citing this R package in publications.

##

## Attaching package: 'GA'

## The following object is masked from 'package:utils':

##

## de

Rastrigin <- function(x1, x2) {

20 + x1^2 + x2^2 - 10\*(cos(2\*pi\*x1) + cos(2\*pi\*x2))

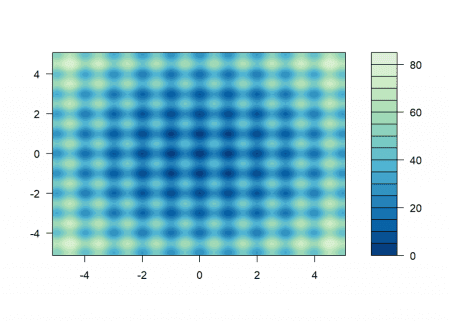
}

x1 <- x2 <- seq(-5.12, 5.12, by = 0.1)

f <- outer(x1, x2, Rastrigin)

persp3D(x1, x2, f, theta = 50, phi = 20)

filled.contour(x1, x2, f, color.palette = bl2gr.colors)



To find the global minimum (spoiler: it is at (0,0)) we use the GA package (because GA only maximizes we use the minus sign in front of the fitness function):

set.seed(70)

GA <- ga(type = "real-valued",

fitness = function(x) -Rastrigin(x[1], x[2]),

lower = c(-5.12, -5.12), upper = c(5.12, 5.12),

maxiter = 1000)

summary(GA)

## -- Genetic Algorithm -------------------

##

## GA settings:

## Type = real-valued

## Population size = 50

## Number of generations = 1000

## Elitism = 2

## Crossover probability = 0.8

## Mutation probability = 0.1

## Search domain =

## x1 x2

## lower -5.12 -5.12

## upper 5.12 5.12

##

## GA results:

## Iterations = 1000

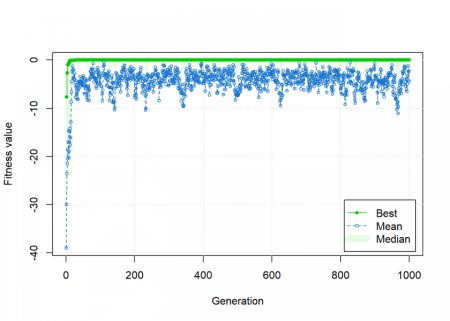
## Fitness function value = -3.630204e-07

## Solution =

## x1 x2

## [1,] 2.81408e-05 3.221658e-05

plot(GA)

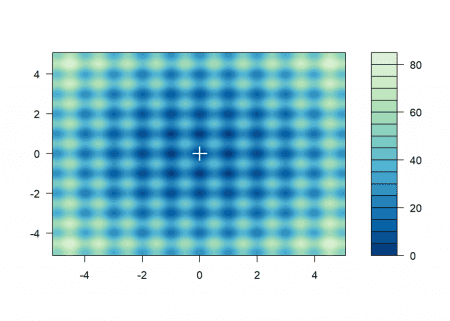


filled.contour(x1, x2, f, color.palette = bl2gr.colors, plot.axes = {

axis(1); axis(2); points(GA@solution[ , 1], GA@solution[ , 2], pch = 3, cex = 2, col = "white", lwd = 2)

}

)



Quite impressive, isn’t it! Evolution just works!

In an upcoming post we will use evolutionary methods to find a nice functional form for some noisy data with a method called *symbolic regression* or *genetic programming* – so stay tuned!