Genetic Optimization Algorithm

Developing a Basic Genetic Optimization Algorithm in C

Alexis Fernandez

1 Tables

Table 1: Results with Crossover Rate = 0.5 and Mutation Rate = 0.05

Pop Size	Max Gen	Best Solution			CPU time (Sec)
		x_1	x_2	Fitness	
10	100	0.1255521854039063	0.5822782197884644	3.4130903211908437	0.000406
100	100	0.0541874324223901	0.0350230583152840	0.2902269016791874	0.002549
1000	100	-0.0226526730799357	0.0080152764022419	0.0832715336057217	0.013433
10000	100	0.0036207679675986	0.0058276159715973	0.0206583927195578	0.122783
1000	1000	0.0080712419040836	0.0062397681205715	0.0316249934816430	0.132467
1000	10000	0.0005056150259941	0.0028410204699449	0.0083836048957946	1.318943
1000	100000	0.0007266481410371	0.0007121101956402	0.0029052269024565	13.224065
1000	1000000	0.0000329455360930	0.0002647610382480	0.0007565282169826	137.485628

Table 2: Results with Crossover Rate = 0.5 and Mutation Rate = 0.2

Pop Size	Max Gen	Best Solution			CPU time (Sec)
		x_1	x_2	Fitness	
10	100	0.2825222747784677	0.0294541707399549	2.0288024540356351	0.000550
100	100	0.0252815801768014	0.0148037658141940	0.1055826562330329	0.002362
1000	100	0.0158117548636216	0.0014792429290154	0.0516194699066870	0.014613
10000	100	0.0008809589784970	0.0022350368100383	0.0069486638677145	0.130222
1000	1000	0.0017486605801382	0.0003302120605158	0.0051176977394118	0.145214
1000	10000	0.0002311076969983	0.0028676027445442	0.0083574796615022	1.440968
1000	100000	0.0000024819746622	0.0004365434872158	0.0012398261269415	14.370251
1000	1000000	0.0003478536383934	0.0002108863555881	0.0011549721341848	149.956569

2 How to run Makefile

2.1 Report and `Makefile`

The Makefile was created in order to compile the program and link it to the header files and the files needed for the initialization of the functions used in the main function.

In order to run the makefile, first the all of the files need including the makefile need to be in the same directory. After, one can type in the terminal the command "make". Once the makefile is run, the code will be compiled with the specified flags and can then be executed like normal. The makefile can also be cleaned with the command ”make clean”.

The following is a breakdown of how the Makefile works.

’CC = gcc’ - Defines the compiler that will be used. GCC in this case.

’CFLAGS = -Wall -Wextra -std=c99 - Defines the flags to be used when compiling the code. These flags are used to provide warnings to the user for correct coding practice. -Wall is used to display all of the commonly used warnings. -Wextra is used to display the extra warnings and -stx=c99 is used to specify the C language standard.

’SOURCES = OF.c functions.c GA.c’ - Defines the source files to be used when compiling the program. It is important to note that functions.h is not included because it is already included in the headers of the functions included.

’OBJECTS = (SOURCES:.c=.o) - Generates a list of object files from the files listed in SOURCES. It does this by replacing the .c extension with a .o extension.

’EXECUTABLE = GA’ - Specifies the name of the executable to be used.

’LIBRARIES = -lm’ - specifies that the math library will be linked to the files.

all:(EXECUTABLE): - Sets the target to be used.

(EXECUTABLE):(OBJECTS) (CC) (CFLAGS) (OBJECTS) -o (EXECUTABLE)(LIBRARIES) - Builds the executable using the compiler specified by CC, the warning flags specified by CFLAGS, the object files in OBJECTS, the output flag -o, the executable in EXECUTABLE and the libraries specified in LIBRARIES.

.o: .c (CC) (CFLAGS) -c ¡ -o @ - Pattern rule that builds the object file. Uses the input file (¡) and the ouput file (@)

clean: rm -f (OBJECTS) (EXECUTABLE) - Clean rule. Uses rm to remove the object files specified and the executable. The f flag ensures that the terminal does not prompt for confirmation for this action.