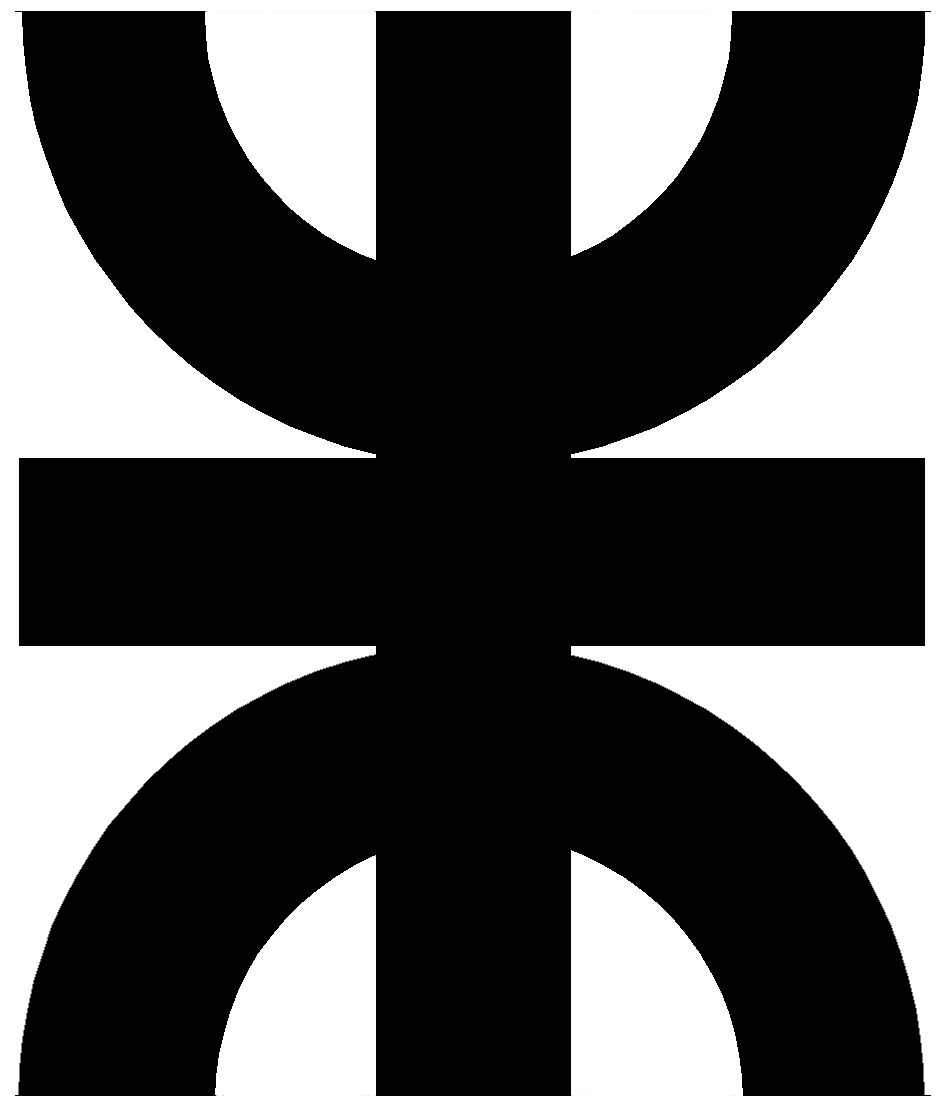
TRABAJO PRÁCTICO Nº 4

Cátedra: Algoritmos Genéticos



**Integrantes Legajo**

Albizuri, Gastón 40412

Belletti, Kristal 40568

Giordano, Nicolás 40467

**UNIVERSIDAD TECNOLÓGICA NACIONAL**

**FACULTAD REGIONAL ROSARIO**

**INGENIERÍA EN SISTEMAS DE INFORMACIÓN**

Índice

**Pág.**

**Introducción** **1**

**Problemática** **2**

**Resolución** **3**

**Código Fuente** **5**

**Resultados**……………………………………………………………………………………………………**17**

**Conclusiones** …**19**

Introducción

*El sudoku es un pasatiempo de ubicación de números en una cuadricula dividida en filas, columnas y subcuadros que visto como un problema de optimización presenta el fenómeno de explosión combinatorial, sus variables no son continuas y en la mayoría de los casos tiene solución única. Estas características lo ubican en la categoría de los problemas NP-Completos.*

*Sudoku es un pasatiempo originario de Estados Unidos se popularizó en Japón en 1986 y se dio a conocer en el ámbito internacional en 2005. El objetivo es llenar la cuadricula de 9x9 celdas (81 celdas), dividida en subcuadros de 3x3, con las cifras del 1 al 9, partiendo con algunos números ya dispuestos en algunas celdas. Además cada número de la solución aparece solo una vez en cada fila, columna y subcuadro, de ahí el “los números deben estar solos” que evoca el nombre del juego.*

Sin importar la cantidad de Sudokus completos que cumplen las restricciones de filas, columnas y subcuadros que existen, la dificultad matemática radica en la exploración combinatorial que presenta un Sudoku con determinada cantidad de números iniciales (pistas), es decir, entre menos pistas tenga el Sudoku, mayor es el espacio de solución (factibles e infactibles) que existen, un Sudoku bien planteado debe tener solución única.

En cuanto a la existencia de solución única, no ha sido posible demostrar cuál es la cantidad mínima de pistas que debe tener un Sudoku para garantizar solución única, aunque es fácil determinar que con 78, 79 y 80 pista se garantiza solución única. Respecto a 77, se encuentran dos posibles soluciones para un Sudoku.

Problemática

Resolver los siguientes casos de Sudoku mediante algoritmo genéticos con los siguientes parámetros:

Población: 16 cromosomas

Ciclos: 100

Tasa de cruzamiento: 90%

Cantidad de mutaciones: 10

Mutación en filas: 10 %

Mutación en columnas: 30 %

Mutación en cuadros: 30 %

Mutación en elementos: 30 %



Resolución

El algoritmo genético en su forma básica aplicado a la solución del Sudoku considera las

siguientes etapas:

**A. Codificación del problema**

Cada configuración del problema es una matriz con números entre 1 y n.

**B. Población inicial**

La población inicial es generada de forma aleatoria. Para cada celda en blanco del problema, se genera un número aleatorio entre 1 y 9. El proceso se repite para cada configuración de la población.

**C. Calculo de la función objetivo**

La función objetivo de cada configuración, debe reflejar la cantidad de números repetidos en cada fila, columna y subcuadro, dado que computacionalmente es ineficiente contar repeticiones, la función objetivo se calculó determinando los números que faltan en filas, columnas o subcuadros (figura 3). La función objetivo de cada configuración es la suma de los faltantes en filas, columnas y subcuadros. De esta forma se determina la mejor configuración encontrada y se almacena como incumbente.

**D. Proceso de selección**

El proceso de selección es proporcional y realizado mediante el método de la ruleta, paraesto se utiliza una función de adaptación que garantiza selectividad y transforma el valorde función objetivo original de minimización a maximización:

Fa = max( Fo ) k - Fo

donde:

Fa: Función de adaptación del individuo i-ésimo.

Fo: Vector función objetivo de los individuos.

Fa: Función objetivo del individuo i-ésimo.

K : Tasa de adaptación (k >1).

**E. Proceso de crossover**

El proceso de crossover se realizó de tal manera que se conservaran las mejores características de los padres. Debido que en el proceso la población se mantiene constante, cada pareja de padres tendrá dos hijos. El primero, tendrá los mejores subcuadros de los padres, es decir, aquellos subcuadros que presentan menos repeticiones; el segundo tendrá las mejores filas o columnas de los dos padres, seleccionado de forma aleatoria con i probabilidad de ocurrencia.

**F. Proceso de mutación**

El proceso de mutación se realizó para generar diversidad ya que la solución normalmente es única y se debe realizar una buena exploración del espacio de soluciones. Cuatro tipos de mutación fueron realizadas, controladas por probabilidades, según el problema: intercambio de dos elementos en una fila, intercambio de dos elementos en una columna, intercambio de dos elementos en un subcuadro y reemplazo de un número del individuo por otro número generado de forma aleatoria entre 1 y n.

Adicionalmente, para no perder el rastro de búsqueda, si en la nueva población no hay un individuo mejor o igual a la incumbente, se reemplaza el peor individuo de la población por la mejor solución encontrada. En caso de que el algoritmo genético no encuentre la solución, el proceso para cuando un número predefinido de iteraciones (generaciones) es alcanzado.

Código fuente

#imports

import random

from random import randint

import copy

import matplotlib.pyplot as plt

#auxiliares

aux = [];

#funciones

def imprimir(individuo):

print("===============================")

for i in xrange(0,9):

print(individuo[i])

print("objetivo: "+str(objetivo(individuo)))

print("===============================")

def poblacionInicial(opc, indiv):

ind = []

for i in xrange(9):

fila = []

for j in xrange(9):

if(indiv[i][j] == 0):

rnd = randint(1,9)

while(rnd in fila):

rnd = randint(1,9)

fila.append(rnd)

else:

fila.append(indiv[i][j])

ind.append(fila)

return(ind)

def objetivo(ide):

#la funcion contar devuelve la cantidad de numeros faltantes de cada (fila|columna|subcuadrado)

contTablero=0

for i in xrange(9):

contTablero = contTablero + contar(ide[i])

#z es numero de columna, i es la fila del tablero

for z in xrange(9):

columna = []

for i in xrange(9):

columna.append(ide[i][z])

contTablero = contTablero + contar(columna)

#z es numero de columna, y es el numero de fila

#xrange(0,7,3): va de 0 a 7 aumentando de a 3 posiciones (i=i+3)

for z in xrange(0,7,3):

for y in xrange(0,7,3):

#getSubCuadrado obtiene cada subCuadrado

contTablero = contTablero + contar(getSubCuadrado(ide,y,z))

return(contTablero)

def contar(elemento):

cont = 9

#solo descuenta una vez por numero, entonces si esta repetido no descuenta

#es una forma de contar repetidos

for i in xrange(1,10):

if(i in elemento):

cont = cont-1

return(cont)

def getSubCuadrado(indv, fila, columna):

#scc = SubCuadradoColumna; scf = SubCuadradoFila

subCuadrado = []

scc=0

scf=0

if(fila<=2):

scf = 0

if(fila<=5):

scf= 3

else:

scf=6

if(columna<=2):

scc = 0

if(columna<=5):

scc= 3

else:

scc=6

for i in xrange(scf,scf+3):

for j in xrange(scc,scc+3):

subCuadrado.append(indv[i][j])

return subCuadrado

#programa Principal

poblacion = []

poblacionInicio = []

poblacionNuevoCiclo = []

mejorIndividuo = []

caso =[]

resultadoInd = []

resultadoObj = []

opc=int(raw\_input('Ingrese el caso Deseado ( 1 o 2 )\n'))

if(opc==1):

caso.append([0,2,4,0,0,7,0,0,0])

caso.append([6,0,0,0,0,0,0,0,0])

caso.append([0,0,3,6,8,0,4,1,5])

caso.append([4,3,1,0,0,5,0,0,0])

caso.append([5,0,0,0,0,0,0,3,2])

caso.append([7,9,0,0,0,0,0,6,0])

caso.append([2,0,9,7,1,0,8,0,0])

caso.append([0,4,0,0,9,3,0,0,0])

caso.append([3,1,0,0,0,4,7,5,0])

else:

caso.append([0,0,0,0,3,0,9,0,1])

caso.append([7,0,1,0,0,0,0,0,0])

caso.append([0,0,0,0,4,0,0,8,0])

caso.append([0,9,0,7,0,2,0,0,0])

caso.append([0,0,0,8,0,0,6,0,0])

caso.append([0,3,0,0,6,0,0,0,5])

caso.append([1,6,8,0,0,4,0,9,0])

caso.append([0,0,0,9,0,0,0,7,0])

caso.append([0,0,4,0,0,0,0,0,0])

iteraciones=int(raw\_input('Ingrese cantidad iteraciones: \n'))

for i in xrange(16):

poblacionInicio.append(poblacionInicial(opc,caso))

poblacion = poblacionInicio

mejorObjetivo=objetivo(poblacionInicio[0])

mejorIndividuo=poblacionInicio[0]

for p in xrange(iteraciones):

fitness= []

objetivos=[]

tot= 0

for i in xrange(16):

tot = tot + 9\*9\*9-objetivo(poblacion[i])

for i in xrange(16):

tot = float(tot)

num = (9\*9\*9-objetivo(poblacion[i]))/tot

fitness.append(num)

#generamos objetivos y fitness

#elitismo

listaOrdenada = []

for i in xrange(16):

for j in xrange(14,i+1,-1):

if(objetivo(poblacion[j]) > objetivo(poblacion[j+1])):

aux = copy.deepcopy(poblacion[j])

poblacion[j] = copy.deepcopy(poblacion[j+1])

poblacion[j+1] = copy.deepcopy(aux)

for i in xrange(4):

poblacionNuevoCiclo.append(poblacion[i])

#tiro la ruleta 8 veces, por pares

for o in xrange(8):

#sumFitness es la posicion de la ruleta

sumFitness = 0

tirada1 = random.random()

tirada2 = random.random()

for n in xrange(16):

if((tirada1 > sumFitness) and (tirada1 <= (sumFitness + fitness[n]))):

elegido1 = n

if((tirada2 > sumFitness) and (tirada2 <= (sumFitness + fitness[n]))):

elegido2 = n

sumFitness = sumFitness + fitness[n]

padre1 = copy.deepcopy(poblacion[elegido1]) #le asigno el indice que fue seleccionado al azar, el cual hace corresponder su cromosoma

padre2 = copy.deepcopy(poblacion[elegido2])

hijo2 = copy.deepcopy(padre2)

hijo1 = []

hijo1 = copy.deepcopy(padre1)

crossRandom = random.random() #genero un nro random para el cross [0,1]

if(crossRandom<= 0.9):

#aplico crossover

#Compara cada subCuadrado de cada padre, y el mejor se guarda en hijo 1

for z in xrange(0,7,3):

for y in xrange(0,7,3):

cant1 = contar(getSubCuadrado(padre1,y,z))

cant2 = contar(getSubCuadrado(padre2,y,z))

if(cant1>cant2):

for i in xrange(z,z+3):

for j in xrange(y,y+3):

hijo1[i][j] = padre1[i][j]

else:

for i in xrange(z,z+3):

for j in xrange(y,y+3):

hijo1[i][j] =padre2[i][j]

#Compara cada fila de cada padre y la mejor la guarda en hijo2

hijo2 = []

for i in xrange(9):

cant1 = contar(padre1[i])

cant2 = contar(padre2[i])

if(cant1>=cant2):

hijo2.append(padre1[i])

else:

hijo2.append(padre2[i])

#mutacion Fila

mutRandom = random.random()

if(mutRandom<=0.1):

fila = randint(0,8)

posx = randint(0,8)

posy = randint(0,8)

if(caso[fila][posx] == 0 and caso[fila][posy] == 0):

aux = hijo1[fila][posx]

hijo1[fila][posx] = hijo1[fila][posy]

hijo1[fila][posy] = aux

mutRandom = random.random()

if(mutRandom<=0.1):

fila = randint(0,8)

posx = randint(0,8)

posy = randint(0,8)

if(caso[fila][posx] == 0 and caso[fila][posy] == 0):

aux = hijo2[fila][posx]

hijo2[fila][posx] = hijo2[fila][posy]

hijo2[fila][posy] = aux

#mutacion columna

mutRandom = random.random()

if(mutRandom<=0.3):

columna = randint(0,8)

posx = randint(0,8)

posy = randint(0,8)

if(caso[posx][columna] ==0 and caso[posy][columna] ==0):

aux = hijo1[posx][columna]

hijo1[posx][columna] = hijo1[posy][columna]

hijo1[posy][columna] = aux

mutRandom = random.random()

if(mutRandom<=0.3):

columna = randint(0,8)

posx = randint(0,8)

posy = randint(0,8)

if(caso[posx][columna] ==0 and caso[posy][columna] ==0):

aux = hijo2[posx][columna]

hijo2[posx][columna] = hijo2[posy][columna]

hijo2[posy][columna] = aux

#mutacion subCuadrado

mutRandom = random.random()

if(mutRandom<=0.3):

subCuadradoX = randint(0,2)\*3 #(0=>0; 1=>3; 2=>6)

subCuadradoY = randint(0,2)\*3 #(0=>0; 1=>3; 2=>6)

#posx1 y las demas son posiciones relativas dentro del cuadrado

posx1 = randint(0,2)

posy1 = randint(0,2)

posx2 = randint(0,2)

posy2 = randint(0,2)

if(caso[subCuadradoX+posx1][subCuadradoY+posy1] == 0 and caso[subCuadradoX+posx2][subCuadradoY+posy2] ==0):

#como son relativas, para llegar a la absoluta tengo que sumarle el cuadrado

aux = hijo1[subCuadradoX+posx1][subCuadradoY+posy1]

hijo1[subCuadradoX+posx1][subCuadradoY+posy1] = hijo1[subCuadradoX+posx2][subCuadradoY+posy2]

hijo1[subCuadradoX+posx2][subCuadradoY+posy2] = aux

mutRandom = random.random()

if(mutRandom<=0.3):

subCuadradoX = randint(0,2)\*3 #(0=>0; 1=>3; 2=>6)

subCuadradoY = randint(0,2)\*3 #(0=>0; 1=>3; 2=>6)

#posx1 y las demas son posiciones relativas dentro del cuadrado

posx1 = randint(0,2)

posy1 = randint(0,2)

posx2 = randint(0,2)

posy2 = randint(0,2)

if(caso[subCuadradoX+posx1][subCuadradoY+posy1] ==0 and caso[subCuadradoX+posx2][subCuadradoY+posy2] ==0):

#como son relativas, para llegar a la absoluta tengo que sumarle el cuadrado

aux = hijo2[subCuadradoX+posx1][subCuadradoY+posy1]

hijo2[subCuadradoX+posx1][subCuadradoY+posy1] = hijo2[subCuadradoX+posx2][subCuadradoY+posy2]

hijo2[subCuadradoX+posx2][subCuadradoY+posy2] = aux

#mutacion de un numero del individuo

mutRandom = random.random()

if(mutRandom<=0.3):

posx = randint(0,8)

posy = randint(0,8)

nuevoNumero = randint(1,9) #este es del 1 al 9 porque son los valores

if(caso[posx][posy] ==0):

hijo1[posx][posy] = nuevoNumero

mutRandom = random.random()

if(mutRandom<=0.3):

posx = randint(0,8)

posy = randint(0,8)

nuevoNumero = randint(1,9)

if(caso[posx][posy] ==0):

hijo2[posx][posy] = nuevoNumero

poblacionNuevoCiclo.append(hijo1)

poblacionNuevoCiclo.append(hijo2)

poblacion = poblacionNuevoCiclo

poblacionNuevoCiclo = []

posPeorObj = -1

peorObjetivo = 0

#aca busco el peor de todos

for i in xrange(16):

if(objetivo(poblacion[i])>= peorObjetivo):

peorObjetivo = objetivo(poblacion[i])

posPeorObj = i

#aca busco el mejor de todos

cambio = 0

mejorObjetivoRepeticion = 9\*9\*9 #lo maximo

for i in xrange(16):

if(objetivo(poblacion[i]) < mejorObjetivo):

mejorObjetivo=objetivo(poblacion[i])

mejorIndividuo = copy.deepcopy(poblacion[i])

cambio = 1

if(objetivo(poblacion[i]) < mejorObjetivoRepeticion):

mejorObjetivoRepeticion=objetivo(poblacion[i])

#imprimo el mejor individuo de cada repeticion

resultadoInd.append(mejorIndividuo)

resultadoObj.append(mejorObjetivoRepeticion)

if(cambio == 0):

poblacion[posPeorObj] = copy.deepcopy(mejorIndividuo)

if(mejorObjetivo == 0):

break;

posicionMax = min(xrange(len(resultadoObj)), key = lambda x: resultadoObj[x])

plt.plot(resultadoObj)

plt.show()

for i in xrange(9):

print(resultadoInd[posicionMax][i])

print(resultadoObj[posicionMax])

#for i in xrange(16):

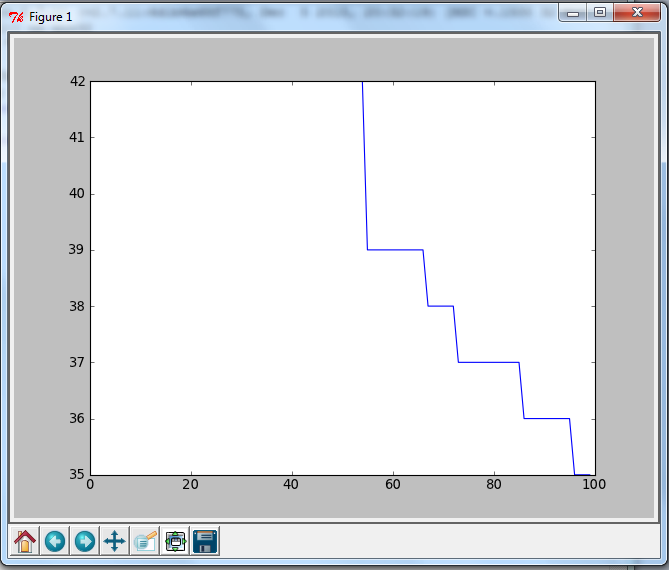
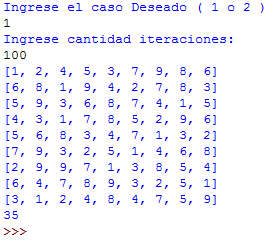
# imprimir(poblacion[i])

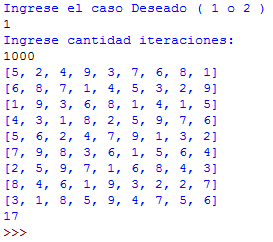
'''nuevaPoblacionBin.append(hijo1) #agrego hijos a la nueva poblacion

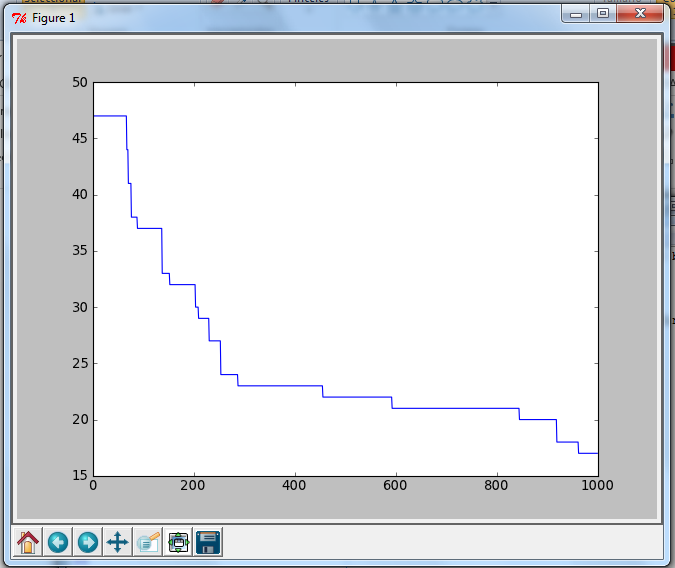
nuevaPoblacionBin.append(hijo2)'''

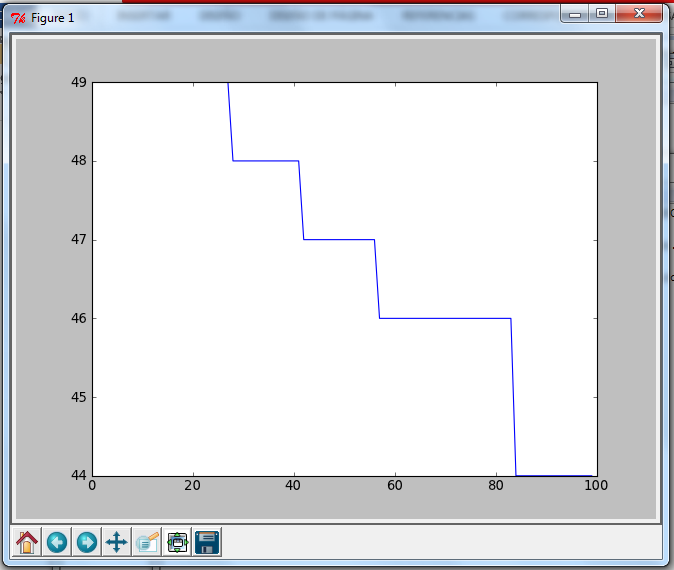
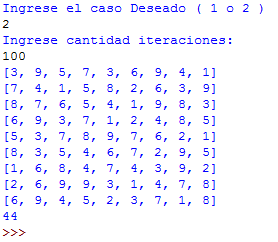
Resultados

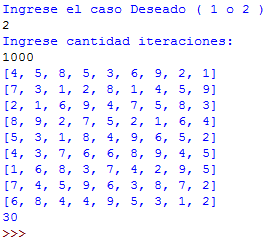
**Generaciones: 100 Caso 1**

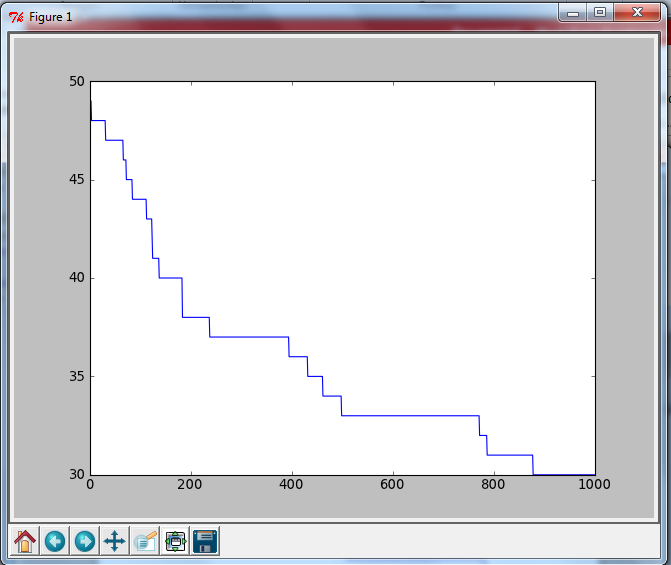


 **Generaciones: 1000**



 **Generaciones: 100 Caso 2**

 **Generaciones: 1000**



Conclusiones

Como se puede ver en los resultados, no es acertado utilizar algoritmos genéticos para encontrar la solución del Sudoku, ya que las posibilidades entre las cuales debe encontrar una solución a un sudoku sencillo son inmensa, y crece exponencialmente cada vez que le quitamos un número.

Se puede ver además que luego de un aumento considerable en la cantidad de iteraciones se ve una leve disminución en el fitness del individuo, por lo que la solución se aproxima luego de una gran cantidad de iteraciones, lo que no es computacionalmente eficiente.