Inteligencia Artificial

TP Nº 2

Algoritmos genéticos

Sergio Piano Padrón: 85191

Mugica Daniel Padrón: 87697

Enunciado:

Realizar una aplicación que permita implementar algoritmos genéticos.

Esta aplicación debe poseer varios parámetros, como el tamaño de la población, los porcentajes de selección, reproducción y mutación a aplicar y la condición de finalización.

El problema a resolver es el siguiente

Obtener las cantidades a fabricar de cada producto de modo de maximizar la ganancia. En la tabla se indican las horas y la cantidad de materia prima necesaria para la fabricación de cada unidad de cada producto. También se

informa la ganancia en pesos obtenida por la fabricación de cada unidad de cada producto.

Producto 1 Producto 2 Producto 3

Horas 6 8 3

Material 4 2 6

Ganancia 7 5 6

Se dispone de 500 Hs y 400 unidades de materia prima.

El TP se puede realizar en cualquier lenguaje.

Se debe entregar el código fuente y gráficos que muestren la evolución de las distintas generaciones.

Introducción

En el presente trabajo práctico se realiza una implementación de Algoritmos Genéticos, en particular se desarrolla un algoritmo genético simple.

Para probar la efectividad del algoritmo se utiliza un modelo lineal, que se busca maximizar. El modelo a resolver es mostrado en el enunciado del trabajo.

El algoritmo recibe parámetros tales como la cantidad de individuos que conforman la población, las probabilidades de que un individuo de la generación siguiente se genere por selección, mutación o reproducción y la cantidad de iteraciones máximas que se pueden realizar.

El proceso de selección se realiza por torneo, es decir, se toman dos individuos al azar y se pasan a la siguiente generación, dos copias del individuo que resulte más apto.

La reproducción se realiza por un punto de cruce. Se toman dos individuos al azar que serán cruzados para obtener un individuo que pasará a la siguiente generación. El cruce se hace tomando un gen al azar, y desde ese gen hacia la izquierda será del padre y hacia la derecha de la madre.

La mutación simplemente lo que hace es alterar un gen tomado al azar y pasar el individuo mutado a la siguiente generación. Este proceso se hace con una probabilidad muy baja.

Código Fuente

**package** app;

**import** geneticos.Evolucion;

**import** geneticos.Fabrica;

**import** geneticos.Individuo;

**import** geneticos.Reproduccion;

**import** geneticos.Seleccion;

**import** geneticos.reproduccion.CruzaSimple;

**import** geneticos.seleccion.Torneo;

**import** java.util.List;

**public** **class** Main {

**private** **static** **int** *sizePoblacion* = 0;

**private** **static** **int** *paramSeleccion* = 0;

**private** **static** **int** *paramReproduccion* = 0;

**private** **static** **double** *tasaMutacion* = 0;

**private** **static** **int** *iteraciones* = 0;

**private** **static** **int** *rangoPoblacion* = 0;

**private** **static** Seleccion *seleccion* = **null**;

**private** **static** Reproduccion *reproduccion* = **null**;

**private** **static** List<Individuo> *poblacionInicial* = **null**;

**public** **static** **void** main(String[] args) {

Main.*init*(args);

*seleccion* = **new** Torneo(*paramSeleccion*);

// poblacion 100; torneo 3; reproduccion 2; mutacion 0.01; iter 10

// poblacion 100; basadoRango 0.4; reproduccion 2; mutacion 0.01; it 10

Evolucion evolucion = **new** Evolucion(*poblacionInicial*, *tasaMutacion*,

*seleccion*, *reproduccion*);

System.*out*.println("N. gen" + ",\t" + "Menor apt" + ",\t" + "Mayor apt" + ",\t" + "Apt promerdio" + ",\t" + "Tam pob" +",\t" + "Cant X1" +",\t" + "Cant X2" +",\t" +"Cant X3" );

// System.out.println("Gen\tMin\tMax\tAvg\tPop");

System.*out*.println(evolucion);

**boolean** continuar = **true**;

**while** (continuar && evolucion.getNumeroGeneracion() < *iteraciones*) {

**try** {

continuar = evolucion.evolucionar();

System.*out*.println(evolucion);

} **catch** (Exception e) {

continuar = **false**;

}

}

System.*out*.println("Poblacion n°: " + evolucion.getNumeroGeneracion());

System.*out*.println("X1" + ",\t" + "X2" + ",\t" + "X3" + ",\t" + "aptitud");

**for**(**int** i= 0 ; i < evolucion.getPoblacion().size() ; i ++){

System.*out*.println(evolucion.getPoblacion().get(i).getCromosoma().get(0)+ ",\t" +evolucion.getPoblacion().get(i).getCromosoma().get(1)+ ",\t" +evolucion.getPoblacion().get(i).getCromosoma().get(2)+ ",\t" +evolucion.getPoblacion().get(i).calcularAptitud());

}

}

**public** **static** **void** init(String[] args) {

Main.*validarCantidadParametros*(args);

Main.*validarEstructuraParametros*(args);

Main.*validarLogicaParametros*();

Main.*configurarComportamientos*();

}

**public** **static** **void** validarCantidadParametros(String[] args) {

**if** (args.length != 6) {

System.*err*.println("Cantidad Incorrecta de Parametros");

Main.*help*();

System.*exit*(1);

}

}

**public** **static** **void** validarEstructuraParametros(String[] args) {

**try** {

*sizePoblacion* = Integer.*parseInt*(args[0]);

*paramSeleccion* = Integer.*parseInt*(args[1]);

*paramReproduccion* = Integer.*parseInt*(args[2]);

*tasaMutacion* = Double.*parseDouble*(args[3]);

*iteraciones* = Integer.*parseInt*(args[4]);

*rangoPoblacion* = Integer.*parseInt*(args[5]);

} **catch** (NumberFormatException e) {

System.*err*.println("Error en el formato de los parametros");

Main.*help*();

System.*exit*(2);

}

}

**public** **static** **void** validarLogicaParametros() {

**if** ( *rangoPoblacion* <= 0

|| *sizePoblacion* <= 0 || *iteraciones* < 0

|| *paramSeleccion* > 100 || *paramSeleccion* <= 0

|| *paramReproduccion* > 31 || *paramReproduccion* < 0

|| *tasaMutacion* > 1 || *tasaMutacion* < 0) {

System.*err*.println("Error logico en los parametros");

Main.*help*();

System.*exit*(3);

}

}

**public** **static** **void** configurarComportamientos() {

*reproduccion* = **new** CruzaSimple(*paramReproduccion*);

*poblacionInicial* = Fabrica.*makePoblacion*(*sizePoblacion*,

*rangoPoblacion*);

}

**public** **static** **void** help() {

System.*out*.println("Parametros:");

System.*out*.println("1ro - Cantidad inicial de la Poblacion (entero)");

System.*out*.println("2ro - Parametro del algoritmo de Seleccion");

System.*out*.println(" (Torneo: Competidores (entero)");

System.*out*.println("3to - Parametro del algoritmo de la Cruza Simple: "

+ "0 - 31 (entero)");

System.*out*.println("4to - Tasa de mutacion de la Poblacion (doble)");

System.*out*.println("5to - Condicion de corte: maximas iteraciones"

+ " (entero)");

System.*out*.println("6vo - Maximo rango de la poblacion inicial");

}

}

**package** app;

**import** java.util.ArrayList;

**public** **class** Funcion {

**private** **static** **final** **int** *GANANCIA\_UNO* = 7;

**private** **static** **final** **int** *GANANCIA\_DOS* = 5;

**private** **static** **final** **int** *GANANCIA\_TRES* = 6;

**public** **int** evaluar(ArrayList<Integer> cromosoma){

**int** ganancia = *GANANCIA\_UNO* \* cromosoma.get(0) + *GANANCIA\_DOS* \* cromosoma.get(1) + *GANANCIA\_TRES* \* cromosoma.get(2);

**return** ganancia;

}

}

**package** geneticos;

**import** java.util.Collections;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.Random;

/\*\*

\* Algoritmo Genetico Simple

\*

\* Un algoritmo genetico consiste en lo siguiente: hallar de que parametros

\* depende el problema, codificarlos en un cromosoma, y se aplican los metodos

\* de la evolucion: seleccion y reproduccion sexual con intercambio de

\* informacion y alteraciones que generan diversidad.

\*

\* **@author** Daniel Mugica

\*/

**public** **class** Evolucion **implements** Iterable<Individuo>{

**private** **int** numeroGeneracion;

**private** **int** contadorReproducciones;

**private** **double** tasaMutacion;

**private** Seleccion seleccion;

**private** Reproduccion reproduccion;

**private** List<Individuo> poblacion;

**public** Evolucion(List<Individuo> poblacionInicial, **double** tasaMutacion,Seleccion seleccion, Reproduccion reproduccion) {

**this**.numeroGeneracion = 0;

**this**.contadorReproducciones = 0;

**this**.tasaMutacion = tasaMutacion;

**this**.seleccion = seleccion;

**this**.reproduccion = reproduccion;

**this**.poblacion = poblacionInicial;

}

**public** **int** getMayorAptitud() {

**return** Collections.*max*(**this**.poblacion).calcularAptitud();

}

**public** **int** getMenorAptitud() {

**return** Collections.*min*(**this**.poblacion).calcularAptitud();

}

**public** Individuo getMejorIndividuo(){

**return** Collections.*max*(**this**.poblacion);

}

**public** **int** getAptitudPromedio() {

**int** suma = 0;

**int** total = 0;

Iterator<Individuo> it = **this**.poblacion.iterator();

**while** (it.hasNext()) {

suma += it.next().calcularAptitud();

total++;

}

**return** suma / total;

}

**public** **int** censarPoblacion() {

**return** **this**.poblacion.size();

}

**public** **int** getNumeroGeneracion() {

**return** **this**.numeroGeneracion;

}

**public** List<Individuo> getPoblacion(){

**return** **this**.poblacion;

}

**public** **boolean** evolucionar() {

**this**.poblacion = **this**.seleccion.seleccionar(**this**.poblacion);

**if** (**this**.poblacion.isEmpty())

**return** **false**;

**this**.entrecruzamiento();

**this**.numeroGeneracion++;

**return** **true**;

}

/\*\*

\* Para aplicar el crossover, entrecruzamiento o recombinacion, se escogen

\* aleatoriamente dos miembros de la poblacion. No pasa nada si se emparejan

\* dos descendiente de los mismos padres; ello garantiza la perpetuacion de

\* un individuo con buena puntuacion (y, ademas, algo parecido ocurre en la

\* realidad; es una practica utilizada, por ejemplo, en la cria de ganado,

\* llamada inbreeding, y destinada a potenciar ciertas caracteristicas

\* frente a otras). Sin embargo, si esto sucede demasiado a menudo, puede

\* crear problemas: toda la poblacion puede aparecer dominada por los

\* descendientes de algun gen, que, ademas, puede tener caracteres no

\* deseados. Esto se suele denominar en otros metodos de optimizacion

\* atranque en un minimo local, y es uno de los principales problemas con

\* los que se enfrentan los que aplican algoritmos geneticos.

\*

\* **@see** http://geneura.ugr.es/~jmerelo/ie/ags.htm

\*/

**private** **void** entrecruzamiento() {

Collections.*shuffle*(**this**.poblacion);

List<Individuo> descendencia = Fabrica.*makePoblacion*();

Iterator<Individuo> it = **this**.poblacion.iterator();

**int** apareados = 0, total = **this**.poblacion.size();

**while** (apareados < (total - total % 2)) {

Individuo padre = it.next();

Individuo madre = it.next();

List<Individuo> hijos = **this**.reproduccion.reproducir(padre, madre);

**this**.contadorReproducciones += hijos.size();

hijos = **this**.mutar(hijos);

descendencia.addAll(hijos);

apareados += 2;

}

**this**.poblacion.addAll(descendencia);

}

/\*\*

\* Una vez establecida la frecuencia de mutacion, por ejemplo, uno por mil,

\* se examina cada bit de cada cadena cuando se vaya a crear la nueva

\* criatura a partir de sus padres (normalmente se hace de forma simultanea

\* al crossover). Si un numero generado aleatoriamente esta por debajo de

\* esa probabilidad, se cambiara el bit (es decir, de 0 a 1 o de 1 a 0). Si

\* no, se dejara como esta. Dependiendo del numero de individuos que haya y

\* del numero de bits por individuo, puede resultar que las mutaciones sean

\* extremadamente raras en una sola generacion.

\*

\*/

**private** List<Individuo> mutar(List<Individuo> poblacion) {

**double** reproducciones = (**double**) **this**.contadorReproducciones;

**double** invMutaciones = 1.0 / **this**.tasaMutacion;

**if** (reproducciones % invMutaciones == 0) {

Iterator<Individuo> it = poblacion.iterator();

**while** (it.hasNext())

**if** ((**new** Random()).nextDouble() < 0.5)

it.next().mutar();

}

**return** poblacion;

}

@Override

**public** Iterator<Individuo> iterator() {

**return** **this**.poblacion.iterator();

}

@Override

**public** String toString() {

Individuo individuo = **this**.getMejorIndividuo();

**return** **this**.getNumeroGeneracion() + ",\t "

+ **this**.getMenorAptitud() + ",\t "

+ **this**.getMayorAptitud() + ",\t "

+ **this**.getAptitudPromedio() + ",\t "

+ **this**.censarPoblacion() + ",\t "

+ individuo.getCromosoma().get(0) + ",\t "

+ individuo.getCromosoma().get(1) + ",\t "

+ individuo.getCromosoma().get(2) + ",\t ";

}

}

**package** geneticos;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Random;

**public** **class** Fabrica {

**public** **static** Individuo makeIndividuo(**int** max) {

ArrayList<Integer> individuo\_parametros = **new** ArrayList<Integer>();

**int** int\_inicial1 = (**new** Random()).nextInt(max);

**int** int\_inicial2 = (**new** Random()).nextInt(max);

**int** int\_inicial3 = (**new** Random()).nextInt(max);

individuo\_parametros.add(0, int\_inicial1);

individuo\_parametros.add(1, int\_inicial2);

individuo\_parametros.add(2, int\_inicial3);

**return** **new** Individuo(individuo\_parametros);

}

**public** **static** List<Individuo> makePoblacion(**int** cantidad,**int** max) {

List<Individuo> poblacion = **new** ArrayList<Individuo>();

**for** (**int** i = 0; i < cantidad; i++)

poblacion.add(Fabrica.*makeIndividuo*(max));

**return** poblacion;

}

**public** **static** List<Individuo> makePoblacion() {

**return** **new** ArrayList<Individuo>();

}

}

**package** geneticos;

**import** java.util.ArrayList;

**import** app.Funcion;

**public** **class** Individuo **implements** Comparable<Individuo>{

**private** ArrayList<Integer> cromosoma;

**private** Funcion aptitud;

**private** **static** **final** **int** *HORAS\_UNO* = 6;

**private** **static** **final** **int** *HORAS\_DOS* = 8;

**private** **static** **final** **int** *HORAS\_TRES* = 3;

**private** **static** **final** **int** *TOTAL\_HORAS* = 500;

**private** **static** **final** **int** *MATERIALES\_UNO* = 4;

**private** **static** **final** **int** *MATERIALES\_DOS* = 2;

**private** **static** **final** **int** *MATERIALES\_TRES* = 6;

**private** **static** **final** **int** *TOTAL\_MATERIALES* = 400;

**public** Individuo(ArrayList<Integer> cromosoma) {

checkIndividuo(cromosoma);

**this**.cromosoma = cromosoma;

**this**.aptitud = **new** Funcion();

}

**private** **void** checkIndividuo(ArrayList<Integer> cromosoma) {

**if**(!(calcularHoras(cromosoma)) && !(calcularMateriales(cromosoma))){

updateCromosoma(cromosoma);

}

}

**private** **void** updateCromosoma(ArrayList<Integer> cromosoma) {

**int** uno = cromosoma.get(0);

**int** dos = cromosoma.get(1);

**int** tres = cromosoma.get(2);

cromosoma.clear();

**if**(uno > dos && uno > tres){

cromosoma.add(0, uno -1);

cromosoma.add(1, dos);

cromosoma.add(2, tres);

}**else** **if**(dos > uno && dos > tres){

cromosoma.add(0, uno);

cromosoma.add(1, dos - 1);

cromosoma.add(2, tres);

}**else**{

cromosoma.add(0, uno);

cromosoma.add(1, dos);

cromosoma.add(2, tres -1 );

}

**if**((calcularHoras(cromosoma)) || (calcularMateriales(cromosoma))){

updateCromosoma(cromosoma);

}

}

**private** **boolean** calcularMateriales(ArrayList<Integer> cromosoma) {

**int** materiales = *MATERIALES\_UNO* \* cromosoma.get(0) + *MATERIALES\_DOS* \* cromosoma.get(1) + *MATERIALES\_TRES* \* cromosoma.get(2);

**if**(materiales > *TOTAL\_MATERIALES*){

**return** **false**;

}

**return** **true**;

}

**private** **boolean** calcularHoras(ArrayList<Integer> cromosoma) {

**int** horas = *HORAS\_UNO* \* cromosoma.get(0) + *HORAS\_DOS* \* cromosoma.get(1) + *HORAS\_TRES* \* cromosoma.get(2);

**if**(horas > *TOTAL\_HORAS*){

**return** **false**;

}

**return** **true**;

}

**public** ArrayList<Integer> getCromosoma() {

**return** **this**.cromosoma;

}

**public** **int** calcularAptitud() {

**return** **this**.aptitud.evaluar(**this**.cromosoma);

}

**public** Funcion getAptitud() {

**return** **this**.aptitud;

}

**public** **void** mutar(){

**int** uno = cromosoma.get(0);

uno ^= (uno+1);

**int** dos = cromosoma.get(1);

dos ^= (dos+1);

**int** tres = cromosoma.get(2);

tres ^= (tres+1);

cromosoma.clear();

cromosoma.add(0, uno);

cromosoma.add(1, dos);

cromosoma.add(2, tres);

}

@Override

**public** **int** compareTo(Individuo individuo) {

**return** **this**.calcularAptitud() - individuo.calcularAptitud();

}

}

**package** geneticos;

**import** java.util.List;

**public** **interface** Reproduccion {

**public** List<Individuo> reproducir(Individuo padre, Individuo madre);

}

**package** geneticos;

**import** java.util.List;

**public** **interface** Seleccion {

**public** List<Individuo> seleccionar(List<Individuo> poblacion);

}

**package** geneticos.reproduccion;

**import** geneticos.Individuo;

**import** geneticos.Reproduccion;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Random;

**public** **class** CruzaSimple **implements** Reproduccion {

**private** **final** **int** puntoDeCruza;

**public** CruzaSimple(**int** puntoDeCruza) {

**if** (!(0 <= puntoDeCruza && puntoDeCruza < Integer.*SIZE*))

**throw** **new** IllegalArgumentException();

**this**.puntoDeCruza = puntoDeCruza;

}

**public** CruzaSimple() {

**this**((**new** Random()).nextInt(Integer.*SIZE* - 1));

}

**private** **int** getPuntoDeCruzaInvertido() {

**return** Integer.*SIZE* - **this**.puntoDeCruza;

}

@Override

**public** List<Individuo> reproducir(Individuo padre, Individuo madre) {

**int** invertido = getPuntoDeCruzaInvertido();

**int** izqPadre1 = izqPadre(padre.getCromosoma().get(0));

**int** izqPadre2 = izqPadre(padre.getCromosoma().get(1));

**int** izqPadre3 = izqPadre(padre.getCromosoma().get(2));

**int** izqMadre1 = izqMadre(madre.getCromosoma().get(0));

**int** izqMadre2 = izqMadre(madre.getCromosoma().get(1));

**int** izqMadre3 = izqMadre(madre.getCromosoma().get(2));

**int** derPadre1 = derPadre(padre.getCromosoma().get(0),invertido);

**int** derPadre2 = derPadre(padre.getCromosoma().get(1),invertido);

**int** derPadre3 = derPadre(padre.getCromosoma().get(2),invertido);

**int** derMadre1 = derMadre(madre.getCromosoma().get(0),invertido);

**int** derMadre2 = derMadre(madre.getCromosoma().get(1),invertido);

**int** derMadre3 = derMadre(madre.getCromosoma().get(2),invertido);

ArrayList<Individuo> hijos = **new** ArrayList<Individuo>(2);

ArrayList<Integer> Elemen\_hijo1 = **new** ArrayList<Integer>();

Elemen\_hijo1.add(0, izqPadre1 << **this**.puntoDeCruza | derMadre1);

Elemen\_hijo1.add(1, izqPadre2 << **this**.puntoDeCruza | derMadre2);

Elemen\_hijo1.add(2, izqPadre3 << **this**.puntoDeCruza | derMadre3);

ArrayList<Integer> Elemen\_hijo2 = **new** ArrayList<Integer>();

Elemen\_hijo2.add(0, izqMadre1 << **this**.puntoDeCruza | derPadre1);

Elemen\_hijo2.add(1, izqMadre2 << **this**.puntoDeCruza | derPadre2);

Elemen\_hijo2.add(2, izqMadre3 << **this**.puntoDeCruza | derPadre3);

Individuo hijo1 = **new** Individuo(Elemen\_hijo1);

Individuo hijo2 = **new** Individuo(Elemen\_hijo2);

hijos.add(hijo1);

hijos.add(hijo2);

**return** hijos;

}

**private** **int** izqPadre(**int** padre) {

**return** padre >> **this**.puntoDeCruza;

}

**private** **int** izqMadre(**int** madre) {

**return** madre >> **this**.puntoDeCruza ;

}

**private** **int** derPadre(**int** padre, **int** invertido) {

**return** (padre << invertido) >>> invertido;

}

**private** **int** derMadre(**int** madre, **int** invertido) {

**return** (madre<< invertido) >>> invertido;

}

}

**package** geneticos.seleccion;

**import** geneticos.Fabrica;

**import** geneticos.Individuo;

**import** geneticos.Seleccion;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Iterator;

**import** java.util.List;

**public** **class** Torneo **implements** Seleccion{

**private** **int** cantidadCompetidores;

**public** Torneo(**int** cantidadCompetidores) {

**if** (cantidadCompetidores < 0)

**throw** **new** IllegalArgumentException();

**this**.cantidadCompetidores = cantidadCompetidores;

}

**public** List<Individuo> seleccionar(List<Individuo> poblacion) {

**int** total = poblacion.size();

**if** (total < 0)

**return** **null**;

Collections.*shuffle*(poblacion);

List<Individuo> seleccionados = Fabrica.*makePoblacion*();

List<Individuo> competidores = **new** ArrayList<Individuo>(

**this**.cantidadCompetidores);

Iterator<Individuo> it = poblacion.iterator();

**int** contador = 0;

**while** (it.hasNext() && (total - contador > **this**.cantidadCompetidores)) {

**for** (**int** i = 0; i < **this**.cantidadCompetidores; i++, contador++){

competidores.add(it.next());

}

seleccionados.add(Collections.*max*(competidores));

competidores.clear();

}

**while** (it.hasNext())

competidores.add(it.next());

seleccionados.add(Collections.*max*(competidores));

**return** seleccionados;

}

}

Resultados y Conclusiones

Se realizaron diferentes corridas variando los parámetros de entrada para poder comparar lo resultados y determinar además el valor máximo del modelo y los parámetros correctos de entrada.

A continuación de muestran los resultados de cada una de las corridas realizadas:

El grafico se interpreta como n° de generación en el eje x y mejor aptitud de esa población en el eje y.

|  |  |  |
| --- | --- | --- |
| Corrida N° 1. Prametros de entrada: | | |
| Tamaño poblacion: | | 100 |
| Parametro torneo: | | 3 |
| Parametro Reproduccion: | | 3 |
| Mutacion: |  | 0.02 |
| N° Iteracion |  | 10 |
| Rango poblacion: | | 31 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| N. gen, | Menor apt, | Mayor apt, | Apt promerdio, | Tam pob, | Cant X1, | Cant X2, | Cant X3 |
| 0, | 69, | 467, | 269, | 100, | 23, | 30, | 26, |
| 1, | 210, | 473, | 356, | 68, | 30, | 19, | 28, |
| 2, | 102, | 473, | 395, | 45, | 30, | 19, | 28, |
| 3, | 372, | 506, | 447, | 29, | 30, | 22, | 31, |
| 4, | 449, | 506, | 475, | 20, | 30, | 22, | 31, |
| 5, | 465, | 506, | 485, | 13, | 30, | 22, | 31, |
| 6, | 498, | 506, | 505, | 9, | 30, | 22, | 31, |
| 7, | 506, | 506, | 506, | 5, | 30, | 22, | 31, |
| 8, | 506, | 506, | 506, | 4, | 30, | 22, | 31, |
| 9, | 506, | 506, | 506, | 4, | 30, | 22, | 31, |
| 10, | 506, | 506, | 506, | 4, | 30, | 22, | 31, |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Corrida N° 2. Prametros de entrada: | | |  |  |  |  |  |
| Tamaño poblacion: | | 100 |  |  |  |  |  |
| Parametro torneo: | | 3 |  |  |  |  |  |
| Parametro  Reproduccion: | | 1 |  |  |  |  |  |
| Mutacion: | | 0.01 |  |  |  |  |  |
| N° Iteracion | | 10 |  |  |  |  |  |
| Rango poblacion: | | 32 |  |  |  |  |  |
| N. gen, | Menor apt, | Mayor apt, | Apt promerdio, | Tam pob, | Cant X1, | Cant X2, | Cant X3 |
| 0, | 77, | 490, | 283, | 100, | 25, | 27, | 30, |
| 1, | 261, | 490, | 371, | 68, | 25, | 27, | 30, |
| 2, | 315, | 490, | 424, | 45, | 25, | 27, | 30, |
| 3, | 401, | 497, | 455, | 29, | 23, | 30, | 31, |
| 4, | 419, | 497, | 481, | 20, | 23, | 30, | 31, |
| 5, | 483, | 505, | 492, | 13, | 27, | 26, | 31, |
| 6, | 483, | 505, | 495, | 9, | 27, | 26, | 31, |
| 7, | 505, | 505, | 505, | 5, | 27, | 26, | 31, |
| 8, | 505, | 505, | 505, | 4, | 27, | 26, | 31, |
| 9, | 505, | 505, | 505, | 4, | 27, | 26, | 31, |
| 10, | 505, | 505, | 505, | 4, | 27, | 26, | 31, |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Corrida N° 3. Prametros de entrada: | | |  |  |  |  |  |
| Tamaño poblacion: | | 100 |  |  |  |  |  |
| Parametro torneo: | | 3 |  |  |  |  |  |
| Parametro Reproduccion: | | 2 |  |  |  |  |  |
| Mutacion: | | 0.01 |  |  |  |  |  |
| N° Iteracion | | 10 |  |  |  |  |  |
| Rango poblacion: | | 31 |  |  |  |  |  |
| N. gen, | Menor apt, | Mayor apt, | Apt promerdio, | Tam pob, | Cant X1, | Cant X2, | Cant X3 |
| 0, | 56, | 478, | 280, | 100, | 30, | 20, | 28, |
| 1, | 200, | 496, | 352, | 68, | 31, | 21, | 29, |
| 2, | 298, | 496, | 413, | 45, | 31, | 21, | 29, |
| 3, | 383, | 511, | 460, | 29, | 30, | 23, | 31, |
| 4, | 448, | 518, | 491, | 20, | 31, | 23, | 31, |
| 5, | 486, | 518, | 502, | 13, | 31, | 23, | 31, |
| 6, | 501, | 518, | 511, | 9, | 31, | 23, | 31, |
| 7, | 518, | 518, | 518, | 5, | 31, | 23, | 31, |
| 8, | 518, | 518, | 518, | 4, | 31, | 23, | 31, |
| 9, | 518, | 518, | 518, | 4, | 31, | 23, | 31, |
| 10, | 518, | 518, | 518, | 4, | 31, | 23, | 31, |

Como puede verse en los resultados anteriores y en sus respectivos gráficos, los parámetros más adecuados son los utilizados en la corrida numero tres. Es importante aclarar que si se hicieran más pruebas con diferentes combinaciones de parámetros posiblemente puede llegar a encontrarse alguna otra configuración mejor.

La combinación encontrada para maximizar las ganancias es la siguiente:

X1 = producto1 = 31 unidades.

X1 = producto1 = 23 unidades.

X1 = producto1 = 31 unidades.

Más allá que el resultado encontrada no es exacto es una aproximación muy buena.

COSAS A MEJORAR:

Se han realizado mas corridas que las expuestas anteriormente, sin embargo algunas de ellas han entregado un valor que no cumple alguna de las condiciones expuestas. La condiciones están restringidas dentro del código expuesto pero es posible que exista algún bag que ante determinadas circunstancias ignora alguna de las restricciones.