Evaluación de Algoritmos Genéticos Aplicados a la Resolución del Problema de Generación de Horarios

Autor:
Daniel Alberto Alegría Sánchez

Asesor Interno:
M.C. Néstor Antonio Morales Navarro

Asesor Externo:
Dr. Noel Enrique Rodríguez Maya

Reporte Final de Residencia Profesional

Instituto Tecnológico de Tuxtla Gutiérrez

Ingeniería en Sistemas Computacionales

2 de Febrero de 2017

Contenido

1 Justificación

La calendarización de horarios es una tarea compleja. Esta tarea implica crear relaciones entre grupos y maestros, después distribuir estas relaciones entre diferentes salones y en diferentes espacios de tiempos. Esto tratando de evitar al máximo las superposiciones entre maestros, grupos y salones de clase.

El realizar este proceso de manera manual, se expone a que el resultado no sea optimo, sin mencionar de que es una tarea tediosa para el o los encargados de realizarla, por lo que lo recomendable es utilizar herramientas que faciliten este proceso, y optimicen el resultado final.

Para resolver este problema, a través de los años los investigadores han abordado varios métodos, como métodos de programación lineal, métodos de coloreo de grafos, métodos de satisfacción de restricciones, métodos de optimización de colonia de hormigas, entre otros [1] [2].

Ademas de los métodos ya mencionados, se encuentran también los métodos de soluciones basados en poblaciones, entre los que se encuentran: algoritmos evolutivos, algoritmos de colonias de abejas, algoritmos de optimización por cúmulo de partículas y algoritmos genéticos.

La generación de horarios utilizando este tipo de herramientas varia en torno a la representación que se le da al problema, por ejemplo, un horario para una universidad es diferente al de una escuela normal, incluso, la forma en que los horarios se representan varian de universidad a universidad. Ciertas universidades crean módulos de cuatro horas por materia, mientras que otras crean módulos de una o dos horas.

Ademas de estas variaciones, se tiene que tomar en cuenta los parámetros de estas herramientas, en el caso de algoritmos genéticos se tiene que contar al menos seis parámetros diferentes. Estos son: numero de población, máximo de generaciones, porcentaje de cruza, porcentaje de muta, método de selección, entre otros.

Este trabajo pretende analizar cada uno de los problemas implicados en la generación de horarios, ademas de evaluar cada uno de los parámetros utilizados por los algoritmos genéticos, esto con la finalidad de encontrar una configuración optima que se adecue a las necesidades del Instituto Tecnológico de Tuxtla Gutiérrez.

2 Objetivos

2.1 Objetivo General

Desarrollar una comparativa entre el rendimiento de diferentes configuraciones de algoritmos genéticos orientados al momento de resolver el problema de generación de horarios de manera que se aprecie las características de cada horario generado con el fin de observar cual configuración es mas optima.

2.2 Objetivos Específicos

- Analizar el estado del arte referente a resolución del problema de horarios utilizando algoritmos genéticos
- Describir detalladamente el problema de generación de horarios
- Identificar métodos de selección y mutación mas utilizados en algoritmos genéticos
- Identificar porcentajes de cruza y mutación mas utilizados en algoritmos genéticos
- Obtener datos escolares para generar horarios
- Construir algoritmos de acuerdo a los datos recolectados
- Recolectar información de cada algoritmo al momento de generar el horario
- Graficar datos
- Crear cuadro comparativo en donde se puedan observar los datos obtenidos

3 Problemática a Resolver

El Instituto Tecnológico de Tuxtla Gutiérrez es una institución educativa que actualmente ofrece nueve licenciaturas, dos maestrías y dos doctorados. Esta institución brinda clases a alrededor de 5000 alumnos, estos están distribuidos en dos turnos: matutino y vespertino. También tiene en su curricula a mas de 200 docentes, cada uno especializado en una cierta area de estudios. Para cubrir las necesidades especificas de ciertas carreras, el ITTG cuenta con laboratorios especializados, como son laboratorios de Física, Química, Mecánica, Eléctrica, entre otros.

Cada semestre se tiene que elaborar un horario para cada grupo de alumnos de esta institución. Este horario debe de tener en cuenta, ademas de las características ya descritas, las preferencias de horarios de los maestros, las restricciones de horarios de los alumnos y las restricciones de cada salon de clases.

El proceso de elaborar estos horarios es una tarea compleja, que aun se hace de manera manual. Debido al número de restricciones, el resultado no siempre es optimo, y es frecuente que existan sobrecargas de trabajo para los profesores, sobrecargas de horas para los alumnos, choques entre horarios, salones no acondicionados para el grupo o materia asignado, etc.

4 Procedimiento y descripción de las actividades realizadas

El desarrollo del proyecto se dividió en cinco partes:

1. Obtención de datos
2. Ordenación de datos
3. Desarrollo del algoritmo genetico
4. Obtención de resultados
5. Presentación de resultados

Hay que indicar que la parte de obtención de datos ya se habia hecho antes de iniciar la residencia profesional. Este semestre se trato de obtener datos actualizados, pero no fue posible.

4.1 Obtención de datos

Par obtener los datos primero se acudió con el jefe de la División de Estudios Superiores, el Ing. Juan José Arreola. Esta división es la que, entre otras actividades, se encarga de de realizar los horarios cada semestre. Para poder entregar los datos requeridos, esta division pidió que se proporcionara un escrito de nuestra parte, en donde se indicase el nombre del proyecto, el nombre de las personas encargadas de desarrollarlo, el objetivo del proyecto y los datos requeridos.

Los datos requeridos eran:

- Lista de maestros (nombres, horas de entrada, horas de salida, materias impartidas)
- Lista de salones (capacidad de alumnos, carrera(s) a la que pertenecen)
- Lista de grupos de alumnos (tamaño, carrera, etc)

Despues de entregar este escrito, la División de Estudios Superiores nos facilito en un archivo los datos que se requerían.

4.2 Ordenación de los datos

El archivo proporcionado por la Division era un horario de un semestre anterior. Este horario era el horario general del ITTG. Lo que se hizo en esta etapa fue extraer unicamente los datos que se necesitaban.

Aunque se pudieron extraer los nombres de los maestros, no hubo forma de obtener sus horarios de entrada y de salida, por lo que lo que se hizo fue obtener la primer y la ultima hora en que se le asignaban clases. Esto nos dio el rango de trabajo de cada maestro.

Aunque los datos ya ordenados contenian informacion de todo el ITTG, para el desarrollo del proyecto se utilizo solamente datos de la carrera de Ingeniería en Sistemas Computacionales.

4.3 Desarrollo del algoritmo genético

El algoritmo fue desrrollado en Java, utilizando MySQL para el manejo de base de datos. Al principio se penso en utilizar la librería ECJ, que es una librería especializada en computo evolutivo, pero debido a que los metodos de seleccion, la mutacion y la cruza ya estaban escritos unicamente Java, esta idea se descarto.

4.3.1 Codificación de los individuos

Para la codificación de los individuos se tuvieron en cuenta varias representaciones. La primera fue de la forma

$$I = \{C, L, T_1 ... T_n, Cl_1 .. Cl_n\}$$

en donde C representa el curso, L el maestro asignado, T el conjunto de horas asignadas y Cl el salon de clases, teniendo en cuenta que para ambos casos, T y Cl, n representa el numero de horas correspondiente al curso.

El principal problema encontrado con esta codificación es que no podría saber con certeza cuando se presentarían choques entre salones, maestros y cursos, por lo que fue descartada.

La segunda codificación que se probo fue una de la forma

$$I = \begin{array}{c|cccc} & T_1 & T_2 & \dots & T_{70} \\ Cl_1 & C,L & C,L & \dots & C,L \\ Cl_2 & C,L & C,L & \dots & C,L \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Cl_n & C,L & C,L & \dots & C,L \end{array}$$

donde T representa a las horas, Cl representa los salones de clases y el conjunto C, L representa a las relaciones Curso-Maestro.

Una explicación detallada aclarando la codificación de los individuos se puede encontrar en 6.1.

4.3.2 Generación inicial de individuos

Después de decidirse la manera en que los individuos se codificaran, se busco la forma en que estos se generaran.

Al principio se opto por una generación totalmente aleatoria. Después de hacer varias pruebas, esta forma de generación se descarto, ya que se perdía mucho tiempo al momento de evaluar los individuos. Ademas, no se podia deliberar si el individuo violaba restricciones flexibles o inflexibles. También surgía un problema al momento de realizar la cruza, ya que la mayoría de las veces los nuevos individuos tenían valores de aptitud peores que los de generaciones anteriores, lo que provocaba que el algoritmo nunca alcanzara valores óptimos.

Debido a esto, se tomo la decision de que los individuos serian horarios validos, es decir, que no violaran ninguna de las restricciones inflexibles, y que el algoritmo genético se encargara de tomar estas soluciones validas y convertirlas a soluciones optimas, tratando de minimizar únicamente las restricciones flexibles, ya que, en un problema con tantas restricciones, se corre el riesgo de crear un algoritmo genético que pase la mayor parte del tiempo evaluando individuos no validos [3].

Un proceso similar se lleva a cabo al momento de generar nuevos individuos a través de la recombinación y la mutación.

4.3.3 Parametros a evaluar

Aunque existen muchos métodos de selección utilizados en los algoritmos genéticos, se decidió evaluar solamente cinco. Estos métodos fueron elegidos ya que se observo que son los mas utilizados para realizar esta tarea. Los métodos son:

- Torneo
- Torneo binario
- Rango lineal
- Ruleta
- Muestreo estocástico universal

Cada algoritmo utiliza una cierta probabilidad de mutación. Los parámetros utilizados para este valor son los siguientes

- 0.1%
- 0.2%
- 0.3%
- 0.4%
- 0.5%
- 0.6%
- 0.7%
- 0.8%
- 0.9%
- 1.0%

Para el numero de generaciones y el tamaño de población se utilizaron 5 valores entre el 100 y el 500.

4.3.4 Restricciones

Restricciones Inflexibles:

- Un maestro, grupo o salon de clases no puede tener superposiciones de cualquier tipo
- Las clases para el turno matutino comienzan a las 7am y terminan, como máximo, a las 4pm
- Las clases para el turno vespertino comienzan, como máximo, a las 12pm y terminan a las 9pm
- Los salones de clases debe de cumplir con los requerimientos de la clase asignada

Restricciones Flexibles:

- Las horas asignadas para cada maestro deben de estar dentro de su horario de trabajo
- Los grupos y maestros no deberán de contener tiempos muertos
- Las clases deberán de estar dispuestas en módulos de dos horas
- El tamaño de los salones debe de coincidir con el tamaño del grupo asignado
- La distribución de las clases entre los salones debe de ser uniforme
- Todos los maestros deben de tener asignados materias

4.3.5 Función Objetivo

Evaluar los individuos que generamos es una tarea critica, ya que de esto depende el resultado final de nuestro algoritmo genético. Debido a que los individuos que generamos no violan las restricciones inflexibles, solo los evaluamos de acuerdo a sus restricciones flexibles.

La función Objetivo se puede representar de la siguiente manera:

$$F = S + O + T + G$$

en donde S se representa como la cantidad de salones cuyo tamaño no coincide con el de la clase asignada.

O es la cantidad de salones cuyo porcentaje de disponibilidad esta muy por debajo o por encima de la disponibilidad media. La disponibilidad de un salon se obtiene de la siguiente manera:

$$D = \frac{70 - clasesAsignadas}{70} * 100$$

De esta manera, si un salon no tiene ninguna clase asignada su disponibilidad sera 100%, por ejemplo.

T requiere de dos valores para calcularlos:

$$T = T_1 + T_2$$

donde T_1 es la cantidad de maestros a los que no se asignaron materias y T_2 es la cantidad de maestros a los que no se les respeto sus horas de trabajo.

G se puede representar como como la desviación estándar de las horas libres. La formula utilizada es:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

donde N es el numero de clases que tiene por días y x es la diferencia de horas que hay entre estas clases. Si el resultado es mayor a 0.5, se aplica la penalización. Por ejemplo:

El grupo S2A tiene 5 clases el día lunes. Estas clases son las siguientes:

Table 1: Ejemplo 1

Materia	Hora
POO	1
POO	2
FISICA	6
FISICA	7

Table 2: Ejemplo 2

Materia	Hora
POO	1
POO	2
FISICA	4
FISICA	5

Las diferencias entre las horas para el primer ejemplo son las siguientes: 1, 4 y 1, por lo tanto, $\sigma = 1.41$. Este caso se penalizaría. Las diferencias entre las horas del segundo ejemplo son 1,2,1, por lo tanto $\sigma = 0.47$. Este caso no se penalizaría ya que, aunque las horas no son consecutivas, solo hay una hora libre.

5 Resultados

6 Anexo

6.1 Individuos

En 4.3.2 se menciono de manera breve la forma en que los individuos se generan. En esta sección se tratara de ampliar esta información.

6.1.1 Inicialización

Antes de asignar horas y salones a las clases, se asignan los maestros. Primero se toman los maestros que pueden dar esa materia, después se elige uno de forma aleatoria:

```
private void initCourses() {
    for (Course course : courses) {
        Gene gene = new Gene.Builder().setCourse(course).
            setTeacher(getRandomTeacher(course)).build();

        for (int i = 0; i < course.getHours(); i++) {
            genes.add(gene);
            gene.getTeacher().addCourse(course);
        }
    }
}
```

Después de asignar un maestro, se crean varias copias de este conjunto. Este numero de copias es dado por la cantidad de horas que la materia tiene asignada:

```
private void arrangeCourses() {
    for (Course course : courses) {
        ArrayList<Gene> courseGenes =
            genes.stream().filter(gene ->
            gene.getCourse().equals(course))
             .collect(Collectors.toCollection(ArrayList<Gene>::new));
        getCoursesPosition(courseGenes);
    }
}
```

6.1.2 Codificación

Cada individuo es representado por un arreglo de dos dimensiones, la dimension de ese arreglo es de $nx70$, en donde n es dado por el numero de salones que ocuparemos. El numero 70 es dado por el numero de horas en las que es posible dar clases. En el ITTG la primer hora de clases es a las 7am, mientras que la ultima hora es a las 9pm. Eso da un total de 14 horas, que multiplicadas por 5, ya que las clases son de lunes a viernes, nos da ese resultado.

Cada posicion de este arreglo se puede definir como una clase en el que no se ha definido ni el curso ni el maestro. Despues de inicalizar el algoritmo, a cada clase se le asigna una

posicion. Este valor representa una posicion dentro de este arreglo. Al asignar esta clase a esa posicion, ese espacio dentro del arreglo deja de estar disponible.

A continuacion se hace una representacion mas grafica de esto.

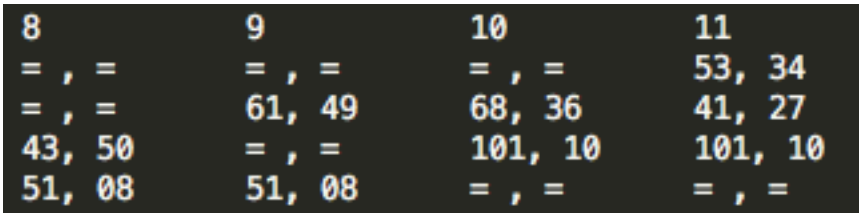


Figure 1: En esta imagen se aprecia un extracto de un individuo. La primer fila de numeros representa la hora y cada fila es un salon. En el caso de la imagen, se representa el rango de horas de 2pm a 6pm para los salones 1, 2, 3 y 4.

	L14-15	L15-16	L16-17	L17-18
Salon1	-	-	-	TBDS5B, T34
Salon2	-	LAUTS6B, T49	GRS8B, T36	TPROGS4B, T27
Salon3	BDDS4B, T50	-	INTARTS9B, T10	INTARTS9B, T10
Salon4	TELCS5B, T08	TELCS5B, T08	-	-

Table 3: Representación del horario generado

6.1.3 Representacion de las horas

La forma en que se representaron las horas fue algo crucial en este trabajo ya que hizo mas facil poder visualizar el numero de horas asignadas a grupos o maestros, ademas de poder apreciar de una mejor manera los posibles choques o violaciones.

La forma en que las horas se representaron fueron con números del 1 al 70, siendo 1 la primer hora del día lunes, y el 70 la ultima hora del día viernes. A continuación se pone el código con los métodos principales de la clase que se utilizo para la representación de las horas.

```
// Time.java
/**
 * La clase Time.java permite manejar de forma mas simple
    nuestras horas. Los valores principales son hour y absHour.
    El valor hour esta representado por un valor del 1 al 14,
    mientras que absHour es un valor entre 1 y 70.
 *
 * @author Alberto Alegria
 */
public class Time {
    private int hour;
    private int absHour;
```

```
private int shift;
private String day;

/*
* Este constructor recibe 1 parámetro que es un numero entre
    1 y 70 representando la hora correspondiente.
*/

public Time(int hour) {
    this.hour = hour % 14;
    if (this.hour == 0) {
        this.hour = 14;     //Esta parte convierte la hora a
            un numero entre 1 y 14. Este numero representa un
            hora pero solo de un dia. Si el numero es 1
            representa al modulo de 7am a 8am. Si es 14
            representa al de 8pm-9pm
    }

    shift = getShift(this.hour);    //Dependiendo del numero
        anterior se calcula a que turno pertenece la hora. Si
        es menor que 7 pertenece al turno matutino, por
        ejemplo.

    day = getDay(((hour - 1) / 14) + 1); //El día esta
        representado por un numero del 1 al 5.
}

public int getShift(int hour) {
    return (hour <= 7) ? Constants.Shift.MORNING :
        Constants.Shift.AFTERNOON;
}

//Retorna true cuando el otro objeto Time esta en el mismo
    día que este.
public boolean isInSameDay(Time anotherTime) {
    return this.day.equals(anotherTime.getDay());
}

//Retorna la diferencia entre horas
public int getHourDifference(Time anotherTime) {
    int difference = this.hour - anotherTime.getHour();

    if (difference < 0) {
        difference = difference * -1;
    }
```

```
        return difference;
    }

    //Retorna la diferencia entre días. Por ejemplo: Si un día
       esta en lunes y el otro en viernes, retorna 4.
    public int getDayDifference(Time anotherTime) {
        int difference = this.dayToInt() - anotherTime.dayToInt();

        if (difference < 0) {
            difference = difference * -1;
        }

        return difference;
    }
   //Retorna la hora entre 1 y 70.
    public int getAbsHour() {
        return hour + (14 * (dayToInt() - 1));
    }

}
```

6.2 Métodos de selección

6.2.1 Torneo

La idea de la selección por torneo es simple. Tomar n individuos de nuestra población, seleccionar al mejor de este grupo y repetir hasta que sea necesario [4].

El numero de individuos puede variar. Para el desarrollo de este trabajo se utilizaron dos tipos de torneo, uno con dos individuos, torneo binario, y otro con cinco individuos.

A continuación se muestran el código de ambos métodos de selección.

```
/*
* Torneo binario
*/
private void binaryTournamentSelection() {
        matingPool.clear();
        for (int i = 0; i < poolSize; i++) {
            Individual individualA =
               population.getIndividual(Methods.getRandomNumber(0,
               population.getSize() - 1));
            Individual individualB =
               population.getIndividual(Methods.getRandomNumber(0,
               population.getSize() - 1));

            if (individualA.getFitness() <
               individualB.getFitness()) {
```

```
                matingPool.add(individualA);
            } else {
                matingPool.add(individualB);
            }
        }
    }

/*
* Torneo de n individuos
*/
private void tournamentSelection() {
        matingPool.clear();

        for (int i = 0; i < poolSize; i++) {
            ArrayList<Individual> competitors = new ArrayList<>();

            for (int j = 0; j < tournamentSize; j++) {
                competitors.add(population.getIndividual(Methods.getRandomNumber(0,
                    population.getSize() - 1)));
            }

            Individual best = competitors.get(0);

            for (int j = 0; j < tournamentSize; j++) {
                if (competitors.get(j).getFitness() <
                    best.getFitness()) {
                     best = competitors.get(j);
                }
            }

            matingPool.add(best);
        }
    }
```

6.2.2 Selección por rangos

La selección por rangos consiste en ordenar la población de acuerdo a su valor de aptitud, y asignar a cada individuo un valor de acuerdo a su nueva posición con respecto a la población. Por ejemplo, si se encuentra en la primer posición su valor sera de 1, en la segunda ser 2, etc. El código utilizado es el siguiente:

```
private void rankSelection() {
        matingPool.clear();
        population.sortByFitness();
        population.reverse();
```

```
        int size = population.getSize();

        for (int i = 1; i <= size; i++) {
            population.getIndividual(i - 1).setRank(i);
        }

        population.shuffle();

        int pool = 0;

        while (pool < poolSize) {
            for (Individual individual :
                population.getIndividuals()) {
                int rand = Methods.getRandomNumber(0,
                    population.getSize());

                if (rand <= individual.getRank() && pool <
                    poolSize) {
                    matingPool.add(individual);
                    pool++;
                }
            }
        }
    }
```

6.2.3 Metodo de la ruleta

Este método de selección da a cada individuo dentro de la población una probabilidad de ser elegido de acuerdo a su valor de aptitud.

Su nombre se debe al hecho de que asignar a los individuos una probabilidad de selección de acuerdo a su aptitud es como asignarles un espacio dentro de una ruleta, entre mas aptos sean, mas espacio tendrán [5].

```
    private void rouletteWheelSelection() {
        matingPool.clear();
        population.sortByFitness();

        population.getIndividuals().stream().forEach(individual ->
                individual.setExpectedFitness((individual.getFitness()
                    == 0) ? 2 : 1 / individual.getFitness()));

        population.reverse();

        double sumFitness = 0.0;
        for (Individual individual : population.getIndividuals())
            {
```

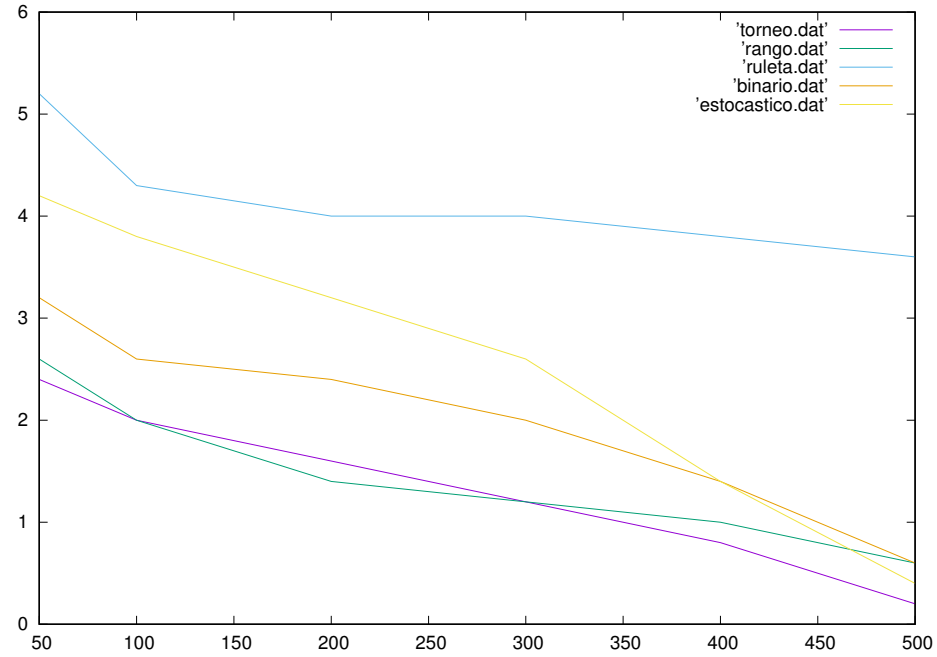
```
            sumFitness += individual.getExpectedFitness();
            individual.setExpectedFitness(sumFitness);
        }

        int pool = 0;
        while (pool < poolSize) {
            double rand = Methods.getRandomDouble(0, sumFitness);

            for (int i = 0; i < population.getSize(); i++) {
                if (rand <=
                    population.getIndividual(i).getExpectedFitness())
                    {
                     matingPool.add(population.getIndividual(i));
                     pool++;
                     break;
                }
            }
        }
    }
```

6.2.4 Muestreo estocástico universal

Este método es una variación del método de la ruleta. Busca ser imparcial, ademas de tener una propagación minima [6].

```
    private void stochasticUniversalSelection() {
        matingPool.clear();
        population.sortByFitness();

        population.getIndividuals().stream().forEach(individual ->
                individual.setExpectedFitness((individual.getFitness()
                    == 0) ? 2 : 1 / individual.getFitness()));

        population.reverse();

        double sumFitness = 0.0;
        for (Individual individual : population.getIndividuals())
            {
            sumFitness += individual.getExpectedFitness();
            individual.setExpectedFitness(sumFitness);
        }

        double distance = sumFitness / poolSize;
        double start = Methods.getRandomDouble(0, distance);

        for (double i = start; i < sumFitness; i += distance) {
```

```
            for (Individual individual :
                population.getIndividuals()) {
                if (i <= individual.getExpectedFitness()) {
                    matingPool.add(individual);
                    break;
                }
            }
        }
    }
```


'torneo.dat'
'rango.dat'
'ruleta.dat'
'binario.dat'
'estocastico.dat'
0
1
2
3
4
5
6
50
100
150
200
250
300
350
400
450
500

References

[1] H. Babaei, J. Karimpour, and A. Hadidi, "A survey of approaches for university course timetabling problem," *Computers and Industrial Engineering*, vol. 86, pp. 43–59, 2015.

[2] A. Schaerf, "A survey of automated timetabling," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 87–127, 1999.

[3] W. Erben and J. Keppler, "A genetic algorithm solving a weekly course-timetabling problem," in *International Conference on the Practice and Theory of Automated Timetabling*. Springer, 1995, pp. 198–211.

[4] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," *Foundations of genetic algorithms*, vol. 1, pp. 69–93, 1991.

[5] K. Jebari and M. Madiafi, "Selection methods for genetic algorithms," *International Journal of Emerging Sciences*, vol. 3, no. 4, pp. 333–344, 2013.

[6] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the second international conference on genetic algorithms*, 1987, pp. 14–21.