UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Optimización para la formación de grupos mediante algoritmos evolutivos

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Emilio Aparicio Rubio
Tutor: Vicente Julián Inglada
Cotutora: Elena del Val Noguera
Cotutor: Juan Miguel Alberola Oltra
2016/2017

Resumen

En esta memoria se pretende explicar el diseño y uso de un algoritmo genético para la generación automática de grupos en el entorno de viajes mediante vehículos. Se propone un modelo para la asignación de viajes a una determinada formación de grupos de viajeros.

El modelo está implementado mediante un algoritmo genético desarrollado en Java, ayudandonos de la librería de algoritmos genéticos JGAP. Los parámetros del modelo son, por parte de los viajeros, el lugar de salida del viajero, la lista de ciudades que no quiere visitar y su presupuesto disponible. Por otro lado, los parámetros del viaje son, el mínimo de asientos disponibles en el vehículo, el máximo asientos disponibles en el vehículo, el lugar de salida del vehículo que va a realizar el viaje, el destino del viaje y el coste de dicho viaje.

Los resultados muestran soluciones válidas que cumplen con todas las restricciones del problema en todos los escenarios propuestos, mejorando considerablemente el tiempo de ejecución de otras aproximaciones basadas en programación lineal.

Summary

In this specification we go to explain the design and use of a genetic algorithm for an automatic generation of groups in the vehicle travel environment.This paper proposes a model for group formation of trips to a certain formation of groups of travelers.

The model is implemented using a genetic algorithm developed in Java, helping us from the library of genetic algorithms JGAP. The parameters of the model are, by the travelers, the place of departure of the traveler, the list of cities that do not want to visit and its budget available. On the other hand, the parameters of the trip are the minimum of seats available in the vehicle, the maximum seats available in the vehicle, the place of departure of the vehicle to be traveled, the destination of the trip and the cost of said trip.

The results show valid solutions that satisfy all constraints of the problem in all the proposed scenarios. This algorithm considerably improves the execution time of other approaches based on linear programming.

Tabla de contenidos

1. Motivación

La idea de conectar a personas que desean viajar juntas surgió en Francia a principios de 2009 gracias al desarrollo de redes sociales mediante internet. La red social denominada Blablacar fue desarrollada en España a partir de 2010 [4] y poco después la red social se ha ido extendiendo por Gran Bretaña, Italia, Portugal, Benelux, Polonia, Alemania, Rusia, Ucrania, India, Croacia, Serbia, Hungría y Rumanía. Uno de los principales motivos para la expansión ha sido el económico al permitir a las personas con poco presupuesto viajar y compartir coche, tras el estallido de la crisis económica en 2008.

El concepto de compartir un vehículo para que el viaje sea más divertido, sociable, y principalmente más económico durante los trayectos, es la principal motivación de este problema.

Por ello, la formación de grupos de personas óptimos para viajar en el vehiculo más adecuado a las condicionas de cada viaje, es un problema con un coste computacional muy elevado si tratasemos de resolver el problema mediante programación lineal, ya que el algoritmo debería calcular todas las combinaciones posibles de personas y viajes hasta encontrar la solución óptima, cosa inviable para un número muy elevado de personas y viajes. Sin embargo, mediante un algoritmo genético evolutivo que mejore la solución en cada iteración se podría resolver dicho problema, ofreciendo soluciones válidas.

Existen diversas aplicaciones actualmente que podrían incluir un algoritmo para sugerir viajes que se adapten a las características del pasajero, tales como Autolib', BlaBlaCar, Car2go, DriveNow, Kandi Technologies, Marshrutka, etc [5]**.** En este tipo de aplicaciones en las cuales el usuario espera una respuesta rápida de la aplicación, indicandole previamente sus características propias, hacen que un algoritmo genético evolutivo encaje perfectamente como herramienta para calcular los viajes que más se adáptan al viajero de una manera rápida y válida.

2. Objetivos

El objetivo principal de este proyecto es desarrollar un algoritmo genético que proponga una solución para la formación de grupos de viajeros que viajen en el vehículo que más se adapta al viaje que se desea realizar. Este algoritmo tendrá en cuenta distintos criterios para la formación de coaliciones.

La elaboración del algoritmo genético debe proporcionar una solución válida que cumpla las restricciones y preferencias de los usuarios.

La función aptitud del problema debe tener en cuenta el lugar de salida tanto del viaje como de los viajeros, el número de plazas disponible en cada viaje, la relación entre el presupuesto del viajero y el coste del viaje, la lista de ciudades en las que el viajero no quiere ir como destino y la buena, mala o indiferente relación existente entre pasajeros dentro del mismo viaje.

Hay que elegir elegir la mejor configuración para distintos tamaños de población y distintos operadores genéticos. En concreto, se deben realizar experimentos en poblaciones desde 300 cromosomas hasta 1500 cromosomas y con los operadores genéticos de cruce y mutación. También se deben realizar experimentos para distinto número de viajeros y distinto número de viajes.

3. Estado del arte

3.1. Algoritmos Genéticos. Fundamentos y Conceptos

Los algoritmos genéticos se inspiran en los procesos de Evolución Natural y Genética [9]. Un algoritmo genético evoluciona a partir de una población de soluciones inicial, intentando producir nuevas generaciones de soluciones que sean mejores que la anterior. El proceso evolutivo es guiado por decisiones probabilísticas (componentes aleatorios) basados en la adecuación de los individuos. La aptitud puede tener una componente heurística. Al final del proceso se espera obtener un buen individuo: **buena solución global al problema**.

Figura 1. Estructura de un algoritmo genético.

La estructura básica de cualquier algoritmo genético, se compone de 3 pasos fundamentales (ver Figuras 1 y 2):

- **Población Inicial**. Es donde generamos nuestra población aleatoria de individuos inicial . Dicha población tendrá una fucnción aptitud inicial asociada.

- **Ciclo generacional**. Son los distintos operadores que utilizamos para modificar el valor de los individuos y crear nuevos individuos(hijos de los anteriores) con una función aptitud que mejora a la de los padres. Los distintos oepradores que existen son **selección**, **cruce**, **mutación** y **reemplazo**.

- **Verificación de la condición de parada**. En cada generación (iteración del algoritmo) seleccionamos el individuo con mayor valor de aptitud(fitness) y volvemos a realizar el ciclo generacional mientras se cumpla nuestra condición de parada del algoritmo (Por ejemplo, un número determinado de iteraciones).

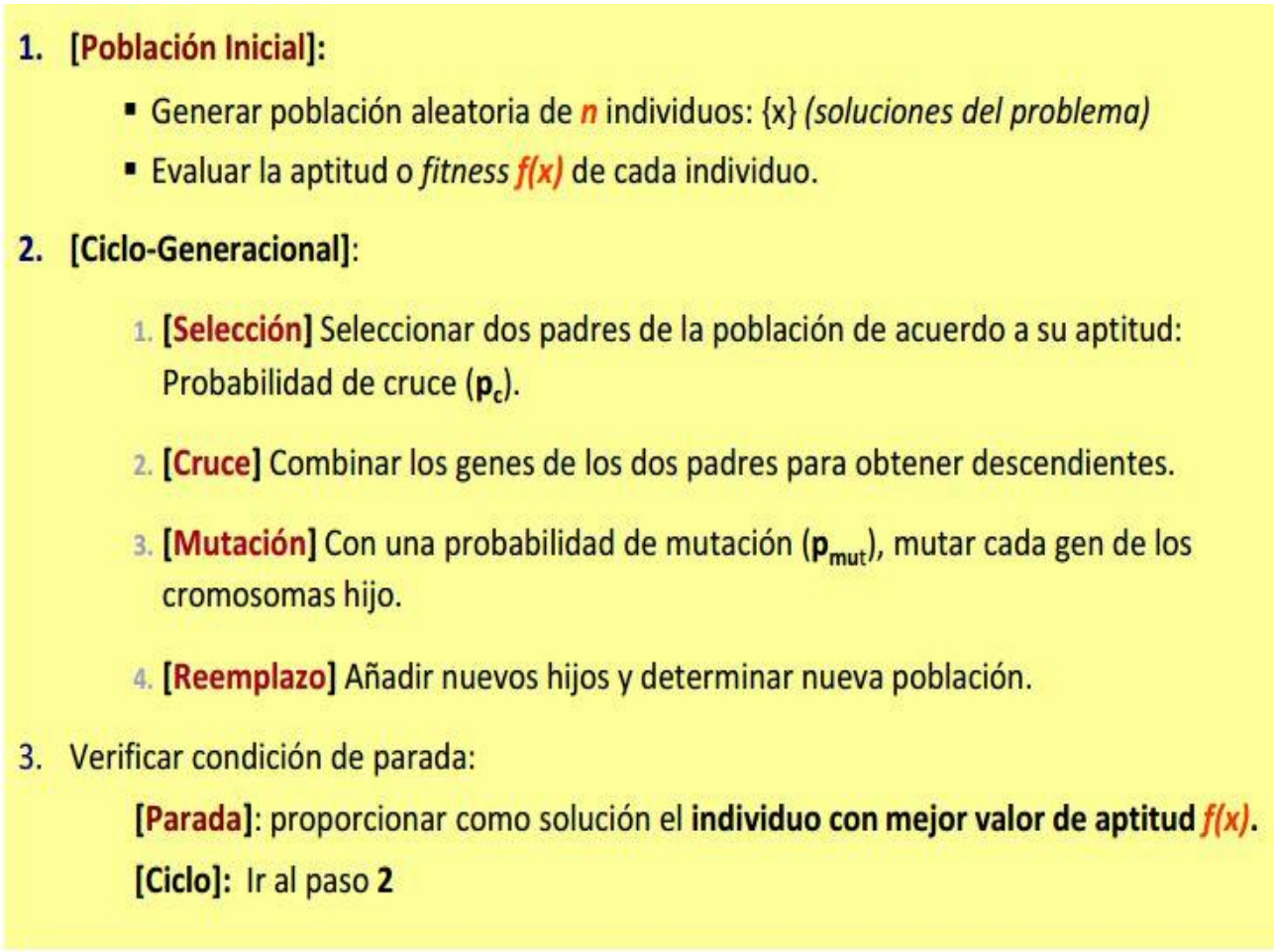

Figura 2. Etapas de un algoritmo genético

3.2. Trabajos anteriores relacionados con el uso de genéticos para generar grupos.

3.2.1. Organización de actividades en residencias para personas mayores.

A continuación se muestra un ejemplo de un trabajo relacionado con la formación de grupos en el entorno de la tercera edad [10].

Diferentes estudios han demostrado los beneficios de la realización de actividades grupales para los ancianos. A menudo se alienta a los miembros de un grupo con capacidades o discapacidades similares a tener la oportunidad de compartir sus experiencias, conocimientos u opiniones. Sin embargo, cuando los cuidadores tratan de planificar actividades de grupo específicas, se deben tener en cuenta diferentes aspectos, como las necesidades a nivel físico, personal, y social de cada persona, lo que aumenta notablemente la complejidad de esa planificación.

Gracias al uso de un algoritmo genético evolutivo, se ha obtenido una herramienta de apoyo a la toma de decisiones para los terapeutas recreativos que facilita la tarea de gestión de agrupar a las personas mayores en grupos óptimos

para la realización de actividades.

3.2.2. Creación de grupos de trabajo en el contexto educativo.

Otro ejemplo de utilización de algoritmo genético, es la formación de grupos en entornos educativos, como puede ser formar grupos de alumnos para distribuirlos en diferentes clases [11].

En este artículo se plantea el uso de programación lineal para encontrar la solución óptima al problema de formación de grupos como primera medida. Posteriormente, se demuestró que temporalmente resulta muy costoso encontrar una solución cuando el número de posibles configuraciones aumenta.

Finalmente se decició plantear un algoritmo genético que no proporcione la solución óptima de los alumnos por grupos, pero si una solución cercana a la óptima que resulta ser una solución válida en un tiempo razonable. El coste temporal de la herramienta mejoró notablemente al aplicar dicho algoritmo genético.

3.3. Aplicaciones existentes en el dominio de aplicaciones colaborativas para viajar en grupo:

Existen varias aplicaciones existentes actualmente que permiten a los usuarios formar grupos de viajeros en un vehículo compartido para realizar un viaje. A continuación se expondrá un breve resumen de tres aplicaciones de dicho dominio:

3.3.1. Blablacar

Blablacar [4] es un servicio francés de vehículo compartido que hace posible que las personas que quieren desplazarse al mismo lugar al mismo momento puedan organizarse para viajar juntos. Permite compartir los gastos puntuales del viaje (combustible y peajes) y también evitar la emisión extra de gases de efecto invernadero, al permitir una mayor eficiencia energética en el uso de cada vehículo.

El portal permite buscar los viajes publicados sin el registro previo; para acordar un viaje se precisa el registro, que es gratuito y que dota al usuario de herramientas para informar sobre sus preferencias de viaje.

Las personas que desean encontrar un coche que comparta el viaje y los gastos, y aquellos que dispongan de un vehículo para compartir, deben registrarse. Tanto el procedimiento de registro como los demás procesos se realizan on-line mediante el uso de aplicaciones para ordenador y smartphone.

La interacción entre las personas que viajan por España es gratuita (mensajes privados y públicos para acordar los detalles del viaje antes de confirmar) pero desde mediados del 2014 las confirmaciones de los viajes se hacen a través del pago on-line. Con este sistema de reserva anticipada los usuarios pueden asegurarse su plaza, también aumenta el compromiso entre las partes, al tiempo que la empresa cobra un porcentaje de cada transacción en concepto de gastos de gestión que varía entre el 10 y el 20%.

3.3.2. Car2go

Car2go [12] es una filial de Daimler AG que proporciona servicios de alquiler de coches en ciudades de Europa y Norteamérica. El servicio de alquiler está descentralizado. El usuario utiliza una aplicación de teléfono inteligente para localizar el coche más cercano y podrá desbloquear la puerta del coche desde la propia aplicación o con una tarjeta de socio. También es posible reservar el vehículo durante un periodo de 20 minutos sin que este tiempo cuente como alquiler.

Ofrece exclusivamente vehículos *Smart Fortwo "car2go edition"*. El alquiler se hace punto a punto y se paga por los minutos usados. Cada ciudad tiene un precio por minuto de alquiler diferente y un usuario registrado podrá utilizar cualquier vehículo car2go del mundo.

En el Departamento de Innovación de Negocio (*Business Innovation department*) en Daimler Jerome Guillen desarrolló el sistema car2go de alquiler de vehículos con las siguientes características:

- No precisa reserva previa.
- Se puede alquilar desde minutos a días.
- Se puede dejar el vehículo aparcado en cualquier plaza de aparcamiento legal.

El sistema se basa en un sofisticado software que empareja coches con conductores sin saber con antelación todos los detalles del alquiler. Además, permite reducir la contaminación en las ciudades y mejorar la movilidad de los usuarios.

3.3.3. Autolib'

Autolib [13] es un servicio de coche eléctrico compartido que se inauguró en París, Francia, en diciembre de 2011. Es un complemento del esquema de bicicletas públicas de la ciudad, Vélib', que fue creado en 2007. El esquema de Autolib tiene la intención de desplegar 3.000 Bluecars Bolloré totalmente eléctricos para uso público mediante una suscripción pagada, basada en una red en toda la ciudad de estacionamiento y estaciones de carga. Autolib' ofrece actualmente más de 4.000 puntos de carga.

Autolib Bluecar está disponible para cualquier persona de 18 o más años, con un permiso de conducir francés válido o una licencia extranjera válida junto con el permiso de conducir internacional, que lleve a cabo una suscripción de pago. Los usuarios pueden elegir entre una serie de paquetes de alquiler, con tasas de 30 minutos que varían de 4 a 8 €, en función del plan de alquiler. Se puede recoger un coche disponible para su uso desde cualquier estación de alquiler y se ha de devolver a cualquier otra estación de alquiler. Cada coche tiene a bordo capacidades GPS y se puede seguir por el centro de operaciones del sistema.

4. Diseño del modelo para la generación de grupos de viajeros.

En este apartado vamos a explicar paso a paso el desarrollo del algoritmo genético para la formación de grupos de viajeros. En un primer momento, describiremos las características primordiales de éste (el algoritmo genético) para, posteriormente, aplicarlo al ámbito que nos ocupa. A continuación, analizaremos el problema en cuestión para luego proponer una solución acorde. Por último, incluiremos modificaciones a fin de afinar y comprobar las soluciones devueltas por el programa.

4.1. Algoritmos genéticos

Un algoritmo genético consta principalmente de cuatro partes:

- **Inicialización**: Generación de la población que consiste en soluciones creadas de forma aleatoria.

- **Evaluación**: Se aplica una función a cada solución que calcula cual buena es ésta.En esta fase también se aplicará algún criterio de detención del algoritmo(condición de parada).

- **Reproducción**: Consiste en aplicar una función de transformación a la población existente. Normalmente esta función consiste en seleccionar dos soluciones de la población y combinarlas, intercambiando aleatoriamente genes entre ellas, por ejemplo. Otro método será la mutación, sobre una solución se genera otro cambiando genes de manera aleatoria.

- **Selección**: Se seleccionan las mejores soluciones para volver al proceso de evaluación. Un criterio sería quedarnos con las mejores soluciones, quedarnos sólo con los hijos, etc…

Más concretamente, el algoritmo que se propone y se explicará a continuación es el siguiente:

```
Generar una población inicial de N cromosomas aleatorios
Evaluar la función aptitud (fitness) de cada cromosoma
Seleccionar la mejor solución s=0
Número de generaciones k=0
Número de generaciones sin mejora de solución q=0
Mientras (k < max_gen ^ q < max_gen sin mejora)
Entonces
Para (j=0; j<N; j++)
 Aplicar aleatoriamente una función de cruce o mutación sobre j
 Evaluar la nueva aptitud (fitness)
 Insertar j’ en la nueva generación
Fin para
 Seleccionar los N mejores
 Seleccionar el mejor s’
 Si (s’<s)
 q++
 Fin si
 k++
Fin Mientras
```

4.2. Especificación del problema

El problema inicial dice así:

"Tenemos un grupo de X viajeros para agruparlas en Y viajes distintos. Cada uno de los viajeros tiene una preferencia para el lugar de salida del viaje, una lista de ciudades que no quiere visitar y un presupuesto disponible. Cada uno de los viajes tiene un mínimo de plazas, un máximo de plazas, un lugar de salida, un destino y un coste asociado. Además, tendremos en cuenta las amistades entre viajeros para la formación de grupos."

Para ilustrar cómo funciona el algoritmo vamos a asumir la situación donde X=16 viajeros e Y=4 viajes, aunque podríamos asumir otras configuraciones de viajeros y viajes.

Los viajes vienen definidos por 6 características, dichas características son el identificador, el mínimo de personas, el máximo de personas, el lugar de salida, el destino y el coste. Las características de los viajes se encuentran en una clase Java denominada Viaje.java (ver Figura 3).

El identificador del viaje es un número que identifica cada viaje. Como existen 4 viajes distintos, será un valor entre el rango 1 al 4 incluidos. Se representa mediante el atributo identificador de la clase Viaje.java y es de tipo Integer.

El mínimo de personas y el máximo de personas es el número de plazas mínimo y máximo, respectivamente, del vehículo que realiza el viaje. Se representan mediante los atributos minPersonas y maxPersonas de la clase Viaje.java y son de tipo Integer.

El lugar de salida es la ciudad origen del viaje. Se representa mediante el atributo lugarSalida de la clase Viaje.java y es de tipo String.

El destino es la ciudad destino del viaje. Se representa mediante el atributo destino de la clase Viaje.java y es de tipo String.

El coste es al cantidad de dinero que cuesta el viaje. Se representa mediante el atributo coste de la clase Viaje.java y es de tipo Integer.

```
public Viaje(int identificador, int minPersonas, int maxPersonas, String lugarSalida, String destino, int coste) {
    this.minPersonas = minPersonas;
    this.maxPersonas = maxPersonas;
    this.lugarSalida = lugarSalida;
    this.destino = destino;
    this.coste = coste;
    this.identificador = identificador;

}
```

Figura 3. Constructor de la clase Viaje.java

Los viajeros vienen definidos por 4 características, dichas características son el identificador, el lugar de salida, la lista de ciudades que no quieres visitar y el coste. Las características de los viajes se encuentran en una clase Java denominada Viajero.java (ver Figura 4).

El identificador del viajero es un número que identifica cada viajero. Como existen 16 viajeros distintos, será un valor entre el rango 0 al 15 incluidos. Se representa mediante el atributo identificador de la clase Viajero.java y es de tipo Integer.

El lugar de salida es la ciudad donde reside el viajero. Se representa mediante el atributo lugarSalida de la clase Viajero.java y es de tipo String.

La lista de ciudades que no quieres visitar es una lista de ciudades en las cuales los viajeros no quieren ir como destino de un viaje asociado. Se representa mediante el atributo ciudadesNoVisitables de la clase Viajero.java y es de tipo String[].

El presupuesto es al cantidad de dinero que tiene el viajero para gastarse en el viaje asociado. Se representa mediante el atributo presupuesto de la clase Viajero.java y es de tipo Integer.

```
public Viajero(int identificador, String lugarSalida, String[]ciudadesNoVisitables, int presupuesto) {
    this.identificador = identificador;
    this.lugarSalida = lugarSalida;
    this.ciudadesNoVisitables = ciudadesNoVisitables;
    this.presupuesto = presupuesto;

}
```

Figura 4. Constructor de la clase Viajero.java

La red social está representada por la matriz de adyacencia *matrizRelacion*, donde el elemento *matrizRelacion[i][j]* toma valores de 0, 1 o 2 si existe una relación entre i y j.

La información relativa a la amistad es un valor fijo que se asume como la relación entre dos viajeros, tomando un valor 0 si el viajero i y el viajero j son enemigos, tomando un valor 2 si el viajero i y el viajero j son amigos y finalmente tomando un valor 1 si el viajero i y el viajero j no se conocen. Por ejemplo, el valor *matrizRelacion[0][9] = 2,* quiere decir que la comparación del viajero que ocupa la posición 0 de la matriz con el viajero que ocupa la posición 9 de la matriz son amigos y, por tanto, posee un valor 2. En nuestro problema, como existen 16 tipos distintos de viajeros, la matriz tendrá una longitud de 16 x 16 celdas, y cada posición contrendrá un valor entero entre 0 y 2 correspondiente a la amistad de comparar el viajero i con el viajero j. Además, la diagonal de la matriz se rellenará con 0, ya que no tiene sentido comparar la amistad de un viajero consigo mismo. Esta matriz es simétrica, por lo tanto, *matrizRelacion[i][j] = matrizRelacion[j][i].*

La siguiente imagen muestra la matriz para el caso del ejemplo que estamos considerando donde había 16 viajeros y 4 viajes.

matrizRelación[Viajero i][Viajero j]

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	1	1	1	1	1	1	1	0	2	0	1	0	1	1	1
1	1	0	2	0	1	1	1	1	1	1	0	1	1	1	1	1
2	1	2	0	1	2	0	1	1	1	1	1	1	1	1	2	1
3	1	0	1	0	2	1	0	1	2	1	1	1	1	1	0	1
4	1	1	2	2	0	1	1	2	0	1	1	2	1	1	1	1
5	1	1	0	1	1	0	1	1	1	2	0	0	1	1	1	1
6	1	1	1	0	1	1	0	1	1	1	1	2	2	0	1	1
7	1	1	1	1	2	1	1	0	1	0	1	1	0	1	2	0
8	0	1	1	2	0	1	1	1	0	1	1	1	1	2	1	1
9	2	1	1	1	1	2	1	0	1	0	1	1	1	2	1	1
10	0	0	1	1	1	0	1	1	1	1	0	1	1	1	1	2
11	1	1	1	1	2	0	2	1	1	1	1	0	1	0	1	1
12	0	1	1	1	1	1	2	0	1	1	1	1	0	1	1	2
13	1	1	1	1	1	1	0	1	2	2	1	0	1	0	1	0
14	1	1	2	0	1	1	1	2	1	1	1	1	1	1	0	1
15	1	1	1	1	1	1	1	0	1	1	2	1	2	0	1	0

Esta matríz se encuentra dentro de la clase funciónAptidud.java y es de tipo Integer[][]. Recalcar que esta matriz se utilizará posteriormente en la función aptitud de nuestro problema.

4.3. Datos de entrada y salida

Los datos de entrada se especificarán en dos ficheros: Viajeros y Viajes.

Un ejemplo de fichero de entrada de Viajeros, será de la siguiente forma:

```
16 ←  Número de viajeros

1     Valencia   [Avila Burgos León Palencia]          500   ← viajero 1
2     Albacete   [Segovia Soria Valladolid Zamora]     300   ← viajero 2
3     Alicante   [A Coruña Bilbao Zaragoza Castellón] 400   ← viajero 3
4     Tarragona  [Jaén Córdoba Huelva León]           450   ← viajero 4

[...]

16    Castellón  [Jaén Bilbao La-Rioja Lugo]         350   ← viajero 15
```

Figura 5. Ejemplo de fichero de entrada de Viajeros al programa

- El primer parámetro de cada viajero será su identificador.
- El segundo parámetro de cada viajero será la ciudad de salida del viajero.
- El tercer parámetro de cada viajero será la lista de ciudades que no quiere visitar. Ejemplo: el viajero 1 no quiere tener un viaje asignado cuyos destinos sean Ávila, Burgos, León o Palencia.
- El cuarto parámetro de cada viajero será el presupuesto disponible.

A continuación se muestra esta información en una tabla para nuestro ejemplo de 16 viajeros. (ver Figuras 5 y 6).

	Identificador	**Lugar de origen**	**Lista de ciudades que no le gustan como destino**	**Presupuesto disponible**
Viajero 1	1	Valencia	Ávila, Burgos, León, Palencia	500 €
Viajero 2	2	Albacete	Segovia, Soria, Valladolid, Zamora	300 €
Viajero 3	3	Alicante	A Coruña, Bilbao, Zaragoza, Castellón	400 €
Viajero 4	4	Tarragona	Jaén, Córdoba, Huelva, León	450 €
Viajero 5	5	Cuenca	Valladolid, Palecia, Valencia, Badajoz	600 €
Viajero 6	6	Ciudad-Real	Guipuzcua, Lugo, Málaga, Cádiz	250€
Viajero 7	7	Murcia	Albacete, Cuenca, Tarragona, Sevilla	200 €
Viajero 8	8	Castellón	Ciudad-Real, Guadalajara, La-Rioja, Cáceres	350 €
Viajero 9	9	Valencia	A coruña, Guipuzcua, Vizcaia, Málaga	535 €
Viajero 10	10	Albacete	Cáceres, Segovia, Valladolid, Tarragona	325 €
Viajero 11	11	Alicante	Valencia, Bilbao, Zaragoza, Castellón	475 €
Viajero 12	12	Tarragona	Jaén, Badajoz, Huelva, León	450 €
Viajero 13	13	Cuenca	Guadalajara, Ciudad-Real, Madrid, Badajoz	625 €
Viajero 14	14	Ciudad-Real	Murcia, Castellón, Málaga, Cádiz	275 €
Viajero 15	15	Murcia	Albacete, Cuenca, Alicante, Ourense	250 €
Viajero 16	16	Castellón	Jaém, bilbao, La-Rioja, Lugo	350 €

Figura 6. Tabla con la composición de los distintos viajeros disponibles

```
4 ←  Número de viajes

1     1     4     Valencia   Madrid     300        ← viaje 1
2     2     4     Albacete   Lugo       250        ← viaje 2
3     1     6     Alicante   Jaen       270        ← viaje 3
4     2     7     Castellón  Palencia   200        ← viaje 4
```

Figura 7. Ejemplo de fichero de entrada de Viajeros al programa

Un ejemplo de fichero de entrada de Viajes, será de la siguiente forma:

- El primer parámetro de cada viaje será su identificador.
- El segundo parámetro de cada viaje será el mínimo de personas.
- El tercer parámetro de cada viaje será el máximo de personas.
- El cuarto parámetro de cada viaje será la ciudad de salida del viaje.
- El quinto parámetro de cada viaje será la ciudad de destino del viaje
- El sexto parámetro de cada viaje será el coste en euros de realizar dicho viaje.

A continuación vamos a mostrar la información de los viajes disponibles en una tabla que muestra la información más claramente: (ver Figuras 7 y 8).

	Identificador	**Mínimo de plazas**	**Máximo de plazas**	**Lugar de origen**	**Lugar de destino**	**Coste del viaje**
Viaje 1	1	1	4	Valencia	Madrid	300 €
Viaje 2	2	2	4	Albacete	Lugo	250€
Viaje 3	3	1	6	Alicante	Jaén	270€
Viaje 4	4	2	7	Castellón	Palencia	200€

Figura 8. Tabla con la composición de los distintos viajes disponibles.

Una vez estipulados los datos de entrada al programa pasamos a recalcar cuales será nuestros datos de salida del programa:

Un ejemplo de impresión por pantalla de resultados de salida és el siguiente:

```
------------------------------------------------------------------------

Mejor individuo total obtenido :
[3,2,3,4,2,1,1,4,4,2,3,4,2,1,1,4] = 3234211442342114

Mejor valor aptitud total obtenido: 540.3214285714286
BUILD SUCCESSFUL (total time: 22 seconds)
```

El mejor individuo total obtenido es el el mejor cromosoma que ha obtenido el algoritmo en la última generación, en este caso és el cromosoma 3234211442342114.

El valor de aptitud obtenido se corresponde a la solución del problema, que será un valor muy cercano a la solución óptima del problema, es este caso, 540.3214285714286.

Por último, la línea BUILD SUCCESSFUL (total time: 22 seconds) indica que el algoritmo ha terminado correctamente en un tiempo total de 22 segundos.

Cabe destacar que el algoritmo también guarda los mejores valores de cada generación en un fichero de salida, para poder guardar los datos y realizar gráficas de resultados.

4.4. Algoritmo para la generación de grupos

Una vez comentado los datos de entrada y salida del programa pasamos a detallar el desarrollo del algoritmo.

Los cromosomas vendrán definidos de la siguiente forma:

0	1	2	3	4	...	12	13	14	15
3	4	1	2	2	...	1	2	4	3

Figura 9. Tabla con la composición de los genes que forman el cromosomas.

La primera fila hace referencia a los identificadores de los viajeros. Por otro lado, la segunda fila hace referencia a los identificadores de los viajes. Así pues, en la primera columna tenemos que el viajero “0” realiza el viaje “3” (ver Figura 9).

Cada cromosoma tendrá un peso que corresponde a su utilidad y vendrá definido de la forma:

$$\sum_{p=0}^{X-1} [\, LugarSalida(p,v) + Presupuesto(p,v) + DestinoGusta(p,v) + PlazasDisponibles(p,v) + Amistad(p,Tv) \,]$$

Siendo:

- “$\boldsymbol{p}$” el número que designa un viajero.
- “$\boldsymbol{v}$” el número de viaje asociado.
- “$\boldsymbol{X}$” el número de viajeros.

- "$\boldsymbol{Tv}$" una matriz de amistades. Esta matriz contendrá la amistad entre un viajero y el resto dentro de un mismo viaje y se corresponderá con la matriz *matrizRelación* comentada anteriormente.

- El método $\boldsymbol{LugarSalida(p,v)}$ premia con un valor 1 a los viajeros cuyo lugar salida coincidan con el lugar de salida del viaje asociado. En caso negativo el valor de aptitud asociado será 0.

- El método $\boldsymbol{Presupuesto(p,v)}$ premia con un valor proporcional presupuesto/coste siempre que el presupuesto del viajero sea inferior o igual al coste del viaje, en caso negativo será un 0.

- El método $\boldsymbol{DestinoGusta(p,v)}$ premia con un valor 1 a los viajeros cuyo viaje no tenga como destino alguna ciudad de la lista de ciudades que no le gusta al viajero asociado. En caso negativo el valor de aptitud asociado será 0.

- El método $\boldsymbol{PlazasDisponibles(p,v)}$ premia con un valor proporcional viajeros/plazas máximas disponibles en un viaje asociado, siempre que el número de viajeros asociados al viaje sea inferior o igual al máximo de plazas dsiponibles. En caso de que el número de viajeros este por encima del máximo de plazas disponibles o por debajo del mínimo de plazas disponibles se le asignará un valor de aptitud 0.

- El método $\boldsymbol{Amistad(p,v)}$ premia con un valor 0, 1 o 2 de aptitud, correspondiente al valor de la *matrizRelación* comentada anteriormente.

Esta función descrita corresponde a la función aptitud del problema y se encuentra en la clase funcionAptitud.java.

Ahora pasaremos a crear los cromosomas iniciales. El número total de cromosomas puede variar, dependiendo de la población que deseemos. Nuestra elección será crearlos de forma aleatoria con valores enteros entre 1 y 4, correspondientes al identificador de los viajes 1 al 4. Además, mostraremos los resultados mediante diferentes configuraciones para comparar los resultados y decidir una buena configuración de operadores genéticos y una buena población inicial.

La condición de parada del algoritmo es de un número de iteraciones = 20.
Esto quiere decir que, a partir de la población inicial del algoritmo mediante un número N de cromosomas aleatorios generados, siendo N la población deseada, el algoritmo efectuará 20 generaciones de cromosomas, mejorando la función aptitud en cada generación gracias a nuestros operadores de cruce y mutación.

En nuestro algoritmo, la función para indicar la población de cromosomas se encuentro en la clase Prueba.java

Por otro lado los operadores genéticos que deciden como efectuar nuevos cromosomas en cada generación son los operadores de cruce y mutación. A continuación pasaremos a detallar el funcionamiento de estos operadores:

1. El **operador de cruce** permite intercambiar aleatoriamente dos genes diferentes dentro de un mismo cromosoma. De este modo, se generará un nuevo cromosoma que será hijo del anterior con dichos genes intercanviados.

0	1	2	3	4	...	12	13	14	15
3	4	1	2	2	...	1	2	4	3

0	1	2	3	4	...	12	13	14	15
3	4	1	1	2	...	2	2	4	3

Figura 10. Tablas con la ilustración del operador genético de cruce.

Como se aprecia, el gen con valor 2 ha intercanviado su posición con el gen con valor 1, es decir, se han "*cruzado*" (ver Figura 10). En nuestro algoritmo el operador de cruce se encuentra en la clase Prueba.java y además, nuestra probabilidad de cruce es de 0,5. Esto quiere decir que cada gen tiene un 50% de probabilidad de intercanviar su posición con el de otro en cada generación de cromosomas.

2. El **operador de mutación** permite cambiar el valor del gen mutado por otro valor aleatorio. De este modo, se generará un nuevo cromosoma que será hijo del anterior con los genes mutados. La mutación se aplica con una probabilidad para cada gen de cada cromosoma.

0	1	2	3	4	...	12	13	14	15
3	4	1	2	2	...	1	2	4	3

0	1	2	3	4	...	12	13	14	15
3	4	1	4	2	...	1	2	4	3

Figura 11. Tablas con la ilustración del operador genético de mutación.

Como se aprecia, el gen con valor 2 ha cambiado su valor y pasado a ser un un gen con valor 4, es decir, ha "*mutado*" (ver Figura 11). En nuestro algoritmo el operador de mutación se encuentra en la clase Prueba.java y además, nuestra probabilidad de mutación es de 0,01. Esto quiere decir que cada gen tiene un 1% de probabilidad de cambiar su valor en cada generación de cromosomas.

5. Experimentación

5.1. Análisis de distintas configuraciones de operadores genéticos

La experimentación ha consistido en una serie de pruebas con los datos de entrada del problema suministrados sobre el algoritmo implementado. Estos datos se han probado para distintas configuraciones que han variado según el número de cromosomas de la población y los operadores genéticos de mutación, de cruce y una combinación de ambos (mutación y cruce juntos).Destacar que dichas pruebas se han realizado para el caso del ejemplo de entrada, donde el número de viajeros es 16 y el número de viajes es 4 , siendo 16 el número de viajeros y 4 el número de viajes.

Las configuraciones escogidas han sido las siguientes:

1. Operador genético de mutación y población inicial de 300 cromosomas.
2. Operador genético de mutación y población inicial de 600 cromosomas.
3. Operador genético de mutación y población inicial de 900 cromosomas.
4. Operador genético de mutación y población inicial de 1200 cromosomas.
5. Operador genético de mutación y población inicial de 1500 cromosomas.
6. Operador genético de cruce y población inicial de 300 cromosomas.
7. Operador genético de cruce y población inicial de 600 cromosomas.
8. Operador genético de cruce y población inicial de 900 cromosomas.
9. Operador genético de cruce y población inicial de 1200 cromosomas.
10. Operador genético de cruce y población inicial de 1500 cromosomas.
11. Combinación de operadores genéticos de cruce y mutación y población inicial de 300 cromosomas.
12. Combinación de operadores genéticos de cruce y mutación y población inicial de 600 cromosomas.
13. Combinación de operadores genéticos de cruce y mutación y población inicial de 900 cromosomas.
14. Combinación de operadores genéticos de cruce y mutación y población inicial de 1200 cromosomas.
15. Combinación de operadores genéticos de cruce y mutación y población inicial de 1500 cromosomas.

A continuación se muestra una tabla de los resultados obtenidos, para la última generación del algoritmo con estas distintas generaciones, siendo el valor de cada celda el resultado obtenido correspondiente a la mayor aptitud obtenida en dicha configuración.

	300	600	900	1200	1500
Mutacíon	538,5714	538,5714	535,8095	536,714	536,8690
Cruce	538,2857	540,2857	540,3214	540,3214	540,3214
Mutación + Cruce	540,2857	540,3214	540,3214	540,3214	540,3214

Figura 12. Tabla de resultados obtenidos.

Donde los números son el número de población de la configuración, mientras que la primera columna corresponde a los distintos operados genéticos de la configuración (ver Figura 12).

A continuación vamos a mostrar los resultados de la tabla en una gráfica, donde el eje de al abscisas representa las distintas configuraciones de población. Irá desde un número de población de 300 cromosomas hasta un número de población de 1500 cromosomas. En el caso de las ordenadas, se representa su valor de utilidad, que irá comprendido desde el valor 533 hasta el valor 541.

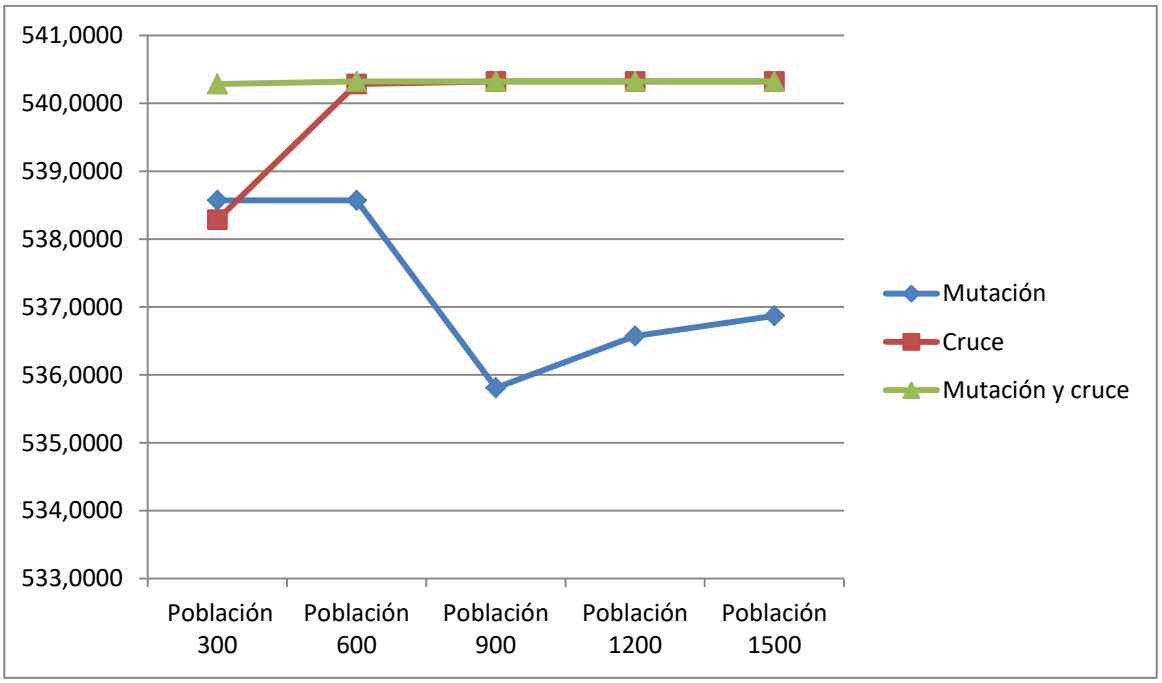

Figura 13. Gráfica población/ valor fitness.

Como se muestra en la gráfica, el operador de mutación con una probabilidad del 50% de cada gen de mutar, no te garantiza mejores soluciones cuando la población aumenta. En este caso, curiosamente con poblaciones con el rango entre 300 y 600 cromosomas es cuando encuentra una solución con una mayor función fitness. Esto es algo que ocurre porque la probabilidad de mutar es muy elevada (ver Figura 13).

Sin embargo, el operador de cruce si que te garantiza que a mayor número de población, mejor valor aptitud obtenido. Cabe destacar que la mejor solución válida del problema en este caso es de un valor de aptitud de 540,3214, por lo que ninguna configuración de cromosomas es capaz de mejorar ese valor. El operador de cruce necesita poblaciones mayores de 600 cromosomas para obtener la mejor solución válida del problema.

Por último, si combinamos nuestro operador genético de cruce con nuestro operador genético de mutación, el algoritmo es capaz de obtener la mejor solución válida en poblaciones muy bajas como se muestra en el gráfico.

A continuación se detalla otro tipo de gráfico, en el que se muestra en distintos bloques los resultados obtenidos, siendo eje de al abscisas los distintos operadores genéticos que estamos comparando y siendo el eje de las ordenadas la suma total del valor de aptitud para los distintos operadores genéticos (ver Figura 14).

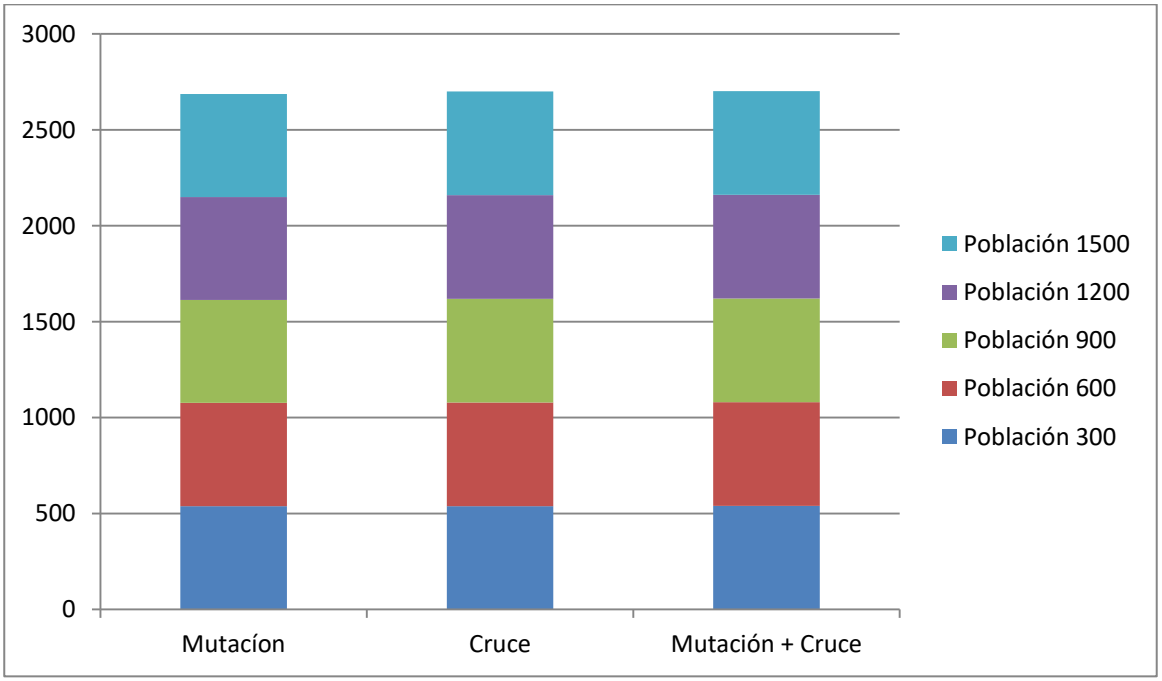

Figura 14. Gráfico de bloques de distintos operadores genéticos.

Para finalizar con la experimentación, se va a mostrar mediante 5 gráficas distintas, la evolución de los resultados en cada iteración del algoritmo. Recordemos que nuestro algoritmo realiza un total de 20 iteraciones y cada iteración será una nueva generación de cromosomas mejor que la anterior, por lo que los resultados siempre van a ser ascendentes en relación con el valor aptitud de cada generación. (ver Figuras 15, 16, 17, 18 y 19).

1. Evolución del valor aptitud para una población de 300 cromosomas.

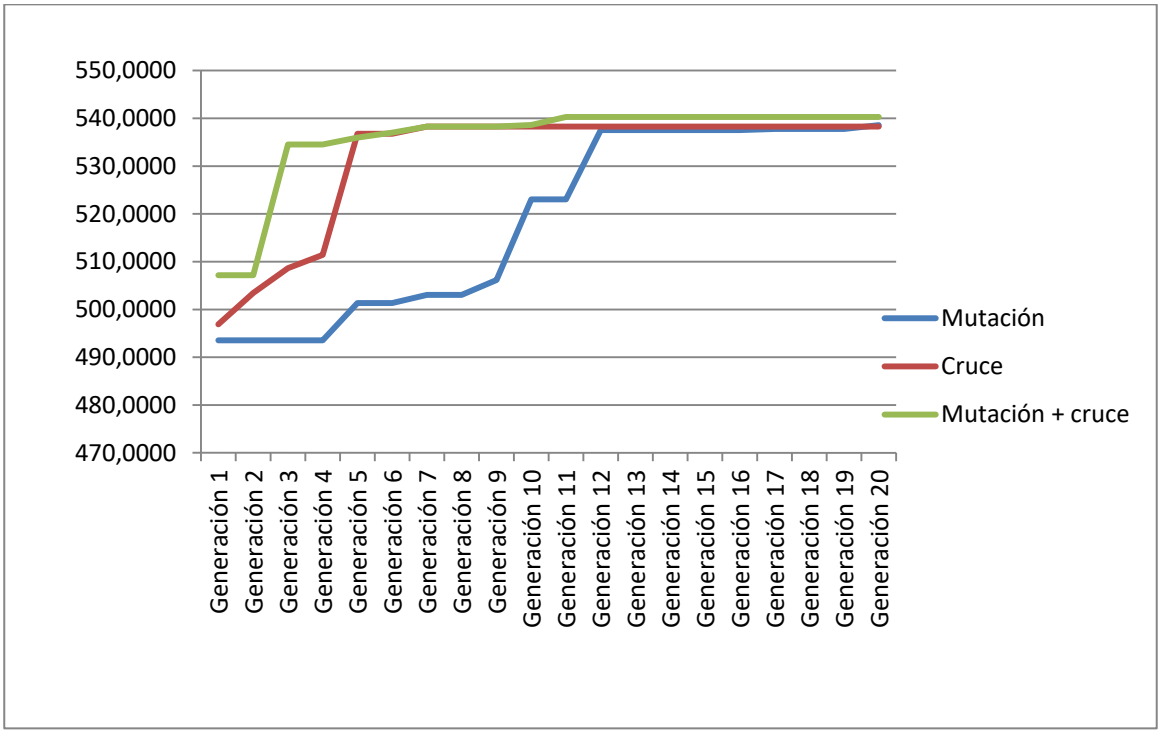

Figura 15. Gráfico de la evolución de la generación para una población de 300 cromosomas.

2. Evolución del valor aptitud para una población de 600 cromosomas.

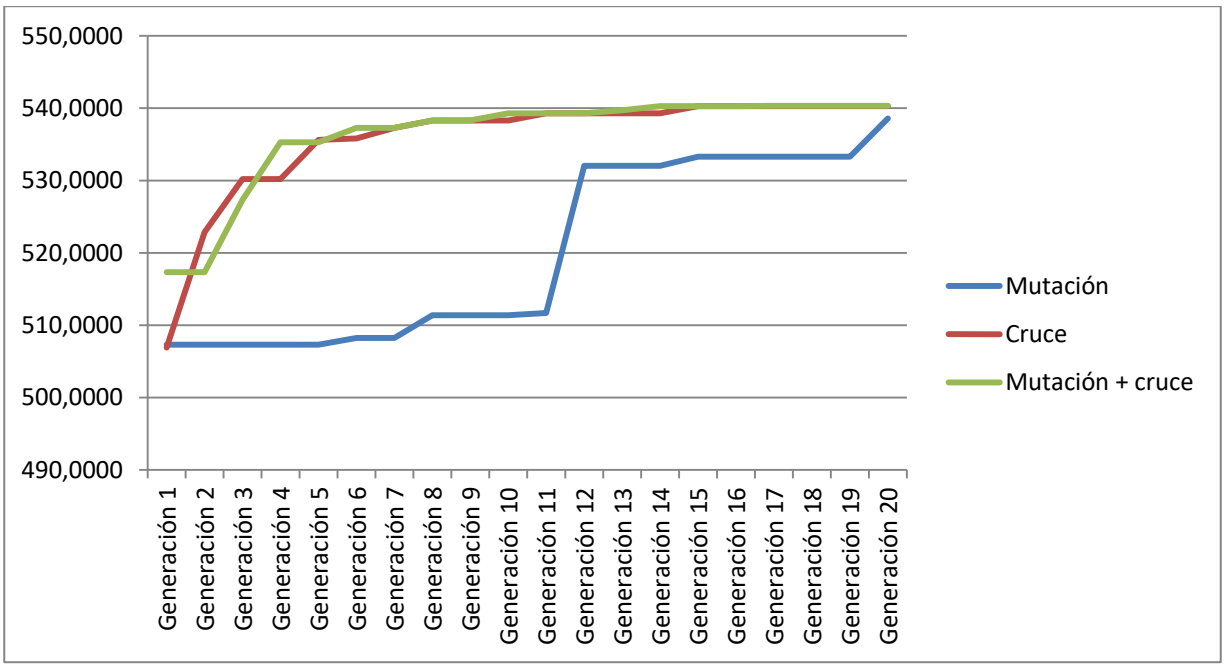

Figura 16. Gráfico de la evolución de la generación para una población de 600 cromosomas.

Evolución del valor aptitud para una población de 900 cromosomas

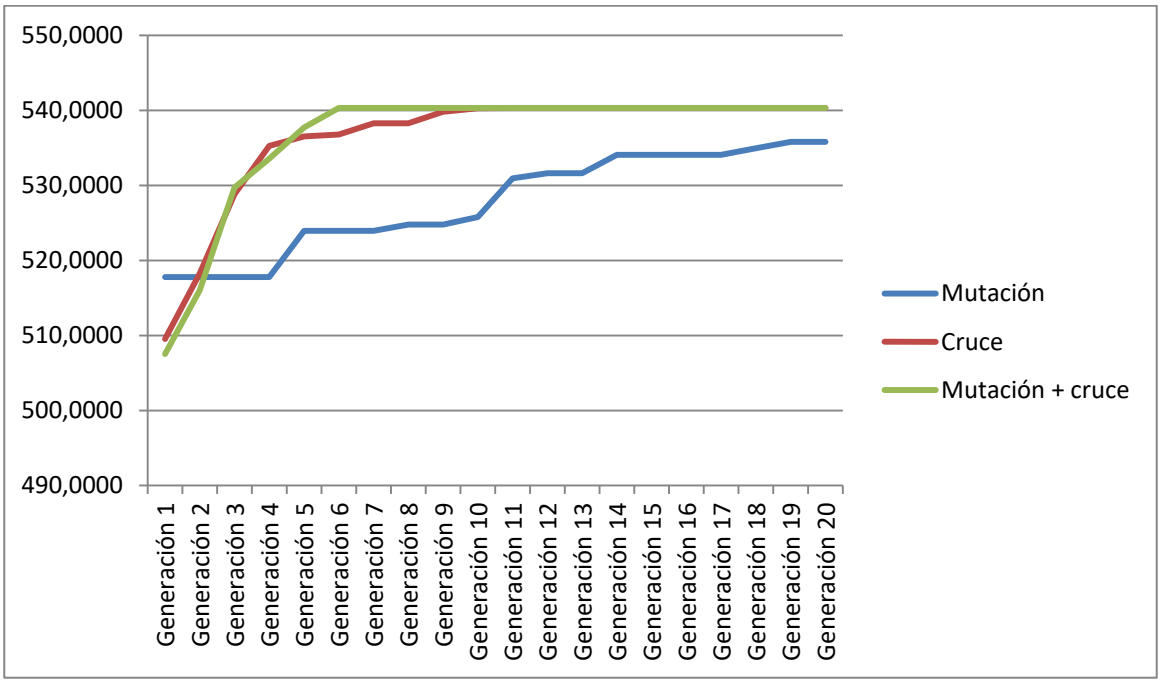

Figura 17. Gráfico de la evolución de la generación para una población de 900 cromosomas.

Evolución del valor aptitud para una población de 1200 cromosomas

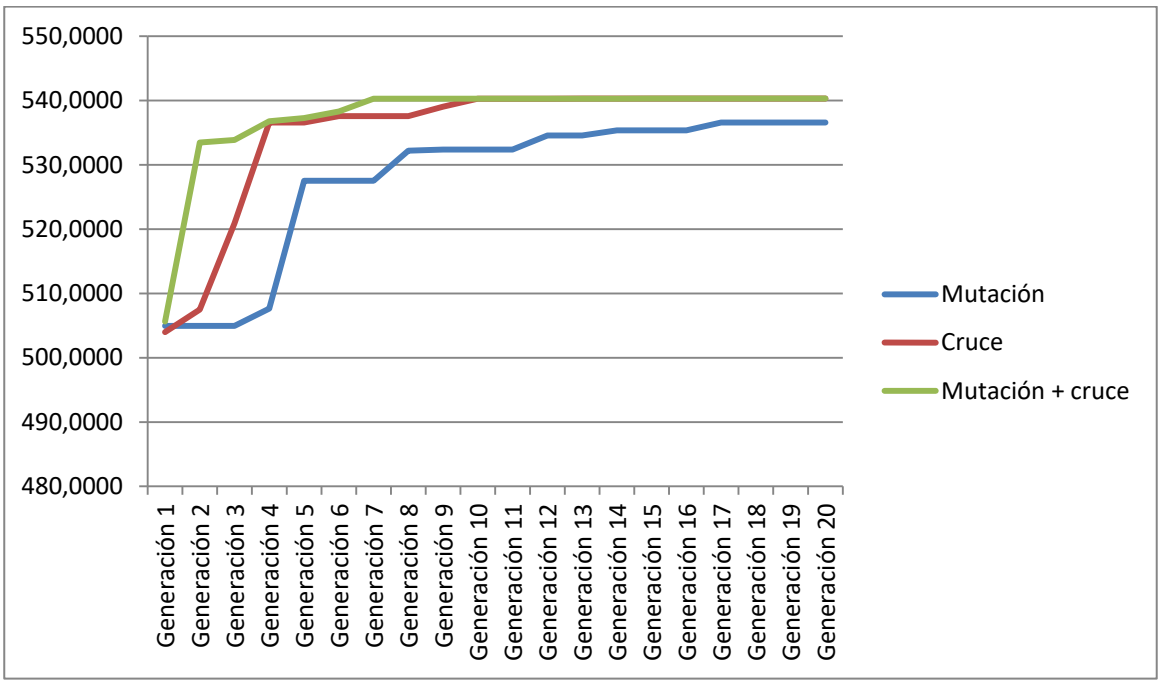

Figura 18. Gráfico de la evolución de la generación para una población de 1200 cromosomas.

Evolución del valor aptitud para una población de 1500 cromosomas

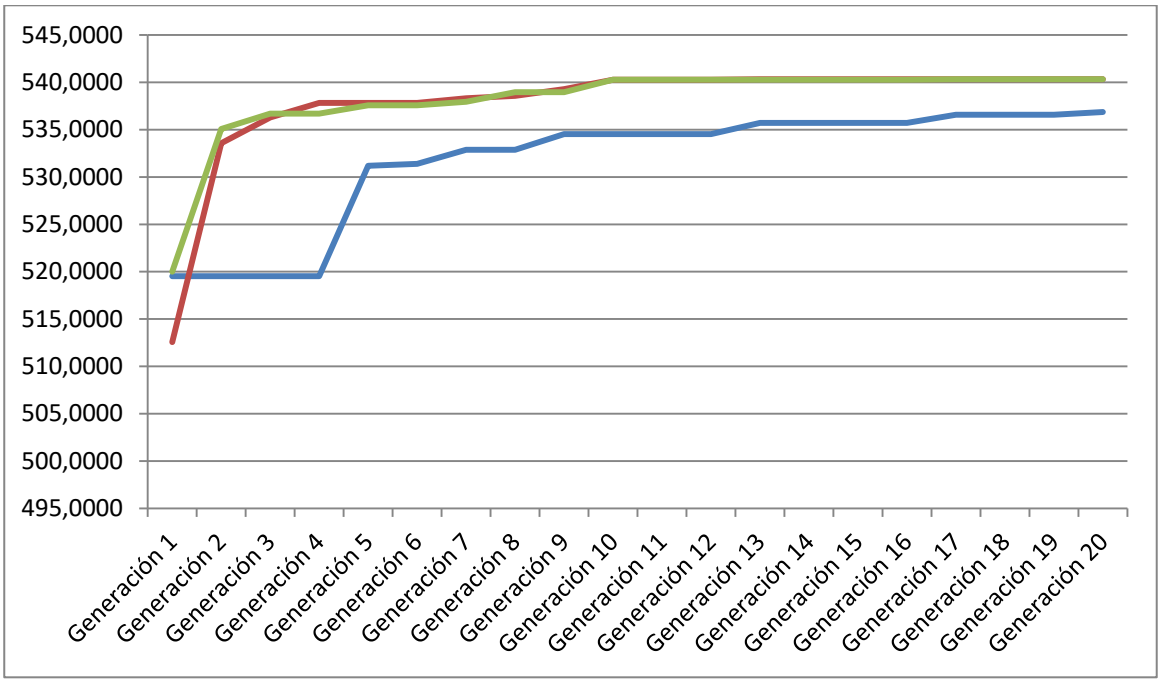

Figura 19. Gráfico de la evolución de la generación para una población de 1500 cromosomas.

A continuación se muestra el resultado del cromosoma cuya combinación de genes tiene la mayor función aptitud posible cumpliendo las restricciones del problema.

```
Mejor individuo total obtenido :
[2,2,3,4,2,1,1,4,4,2,3,4,3,1,1,4] = 2234211442343114

Mejor valor aptitud total obtenido: 540.3214285714286
```

Dicho cromosoma es la mejor solución válida del problema, ya que no hay ninguna otra combinación de genes distinta a la actual que mejore el valor aptitud : **540.3214.**

Además, el algoritmo es capaz de devolvernos también el mejor individuo, que en este caso es el cromosoma cuya combinación de genes es la siguiente: **2234211442343114.**

Si mostramos el resultado en una tabla para que quede mas claro, sería de este modo:

	Viaje 1	**Viaje 2**	**Viaje 3**	**Viaje 4**
Viajero 1		X		
Viajero 2		X		
Viajero 3			X	
Viajero 4				X
Viajero 5		X		
Viajero 6	X			
Viajero 7	X			
Viajero 8				X
Viajero 9				X
Viajero 10		X		
Viajero 11			X	
Viajero 12				X
Viajero 13			X	
Viajero 14	X			
Viajero 15	X			
Viajero 16				X

Figura 20. Tabla ilustrativo del mejor cromosoma del problema.

Donde las X representan que viaje se le ha asignado a cada viajero (ver Figura 20).

5.2. Análisis del algoritmo genético teniendo en cuenta diferentes poblaciones de viajeros

Una vez efectuada las pruebas con los operadores, se ha llegado a la conclusión de que el mejor operador en relación calidad/tiempo es la combinación de operadores de mutación y cruce para una población de 600 individuos.Por ello, ahora vamos a probar con distintas configuraciones de viajes y viajeros de nuestro algoritmo. Se asumirán los resultados para la configuración de :

Población: **600 cromosomas** Operador genético: **mutación y cruce**

5.2.1. Análisis del algoritmo para una población de 8 viajeros

En este apartado vamos a reducir el número de viajeros a 8 para analizar como afecta este cambio a nuestra función aptitud.

En el siguiente gráfico, observaremos como evoluciona nuestra función aptitud a lo largo de las 20 iteraciones de nuestro algoritmo:

Evolución del valor aptitud para una población de 8 viajeros.

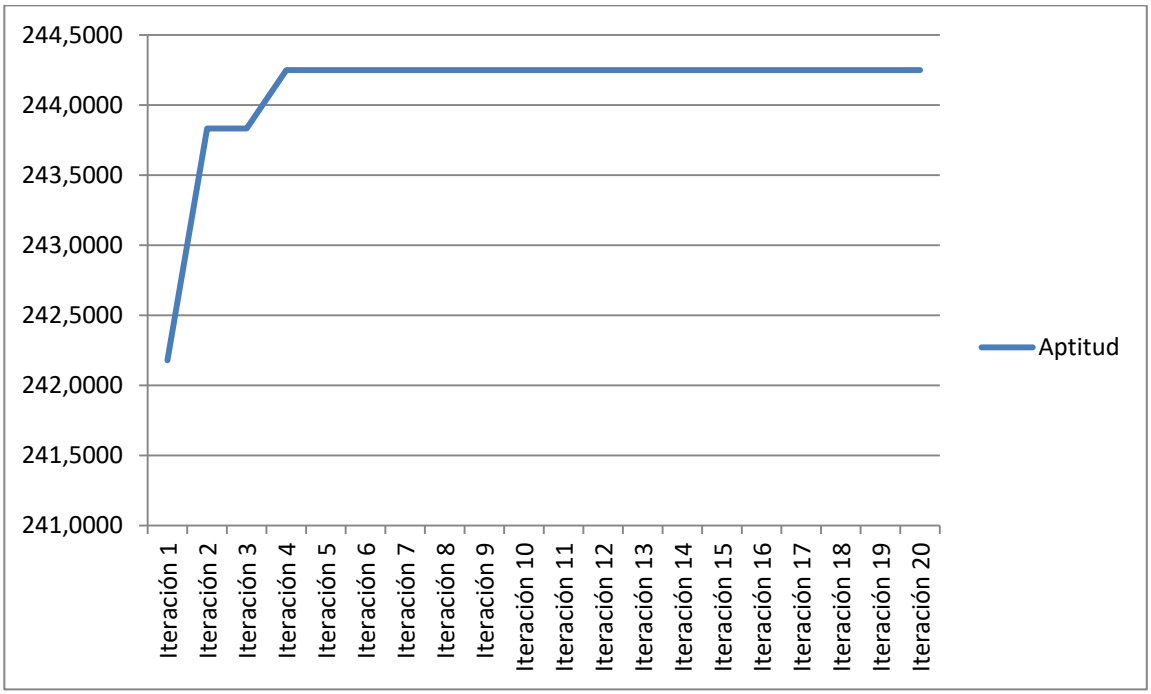

Figura 21. Gráfico de la evolución de la generación para una población de 8 viajeros.

El eje de las abscisas de la gráfica representa la iteración del algoritmo, mientras que el eje de las ordenadas representa el valor de la función aptitud obtenido.

Como se aprecia en la gráfica, al existir menos viajeros, disminuye el número de combinaciones posibles y eso tiene como repercusión que el algoritmo encuentra la máxima solución válida posible muy rápido, en concreto, el algoritmo ha sido capaz de encontrar dicha solución en la generación cuarta de cromosomas(iteración 4 del algoritmo). Se observa que desde la iteracion 4 hasta la iteración 20, el algoritmo no mejora la solución válida obtenida hasta el momento (ver Figura 21).

A continuación se muestra el resultado del cromosoma cuya combinación de genes tiene la mayor función aptitud posible cumpliendo las restricciones del problema.

```
Mejor individuo total obtenido :
[2,2,3,2,2,1,1,4] = 22322114

Mejor valor aptitud total obtenido: 244.25
```

Dicho cromosoma es la mejor solución válida del problema para una población de 8 cromosomas, ya que no hay ninguna otra combinación de genes distinta a la actual que mejore el valor aptitud : **244,25.**

Además, el algoritmo es capaz de devolvernos también el mejor individuo, que en este caso es el cromosoma cuya combinación de genes es la siguiente: **22322114**. A diferencia de la configuración con 16 viajeros, la longitud del cromosoma es de 8 números, siendo cada número el identificador del viaje que se le ha asignado.

5.2.2. Análisis del algoritmo para una población de 32 viajeros

En este apartado vamos a aumentar el número de viajeros a 32 para analizar como afecta este cambio a nuestra función aptitud.

En el siguiente gráfico, observaremos como evoluciona nuestra función aptitud a lo largo de las 20 iteraciones de nuestro algoritmo:

Evolución del valor aptitud para una población de 32 viajeros.

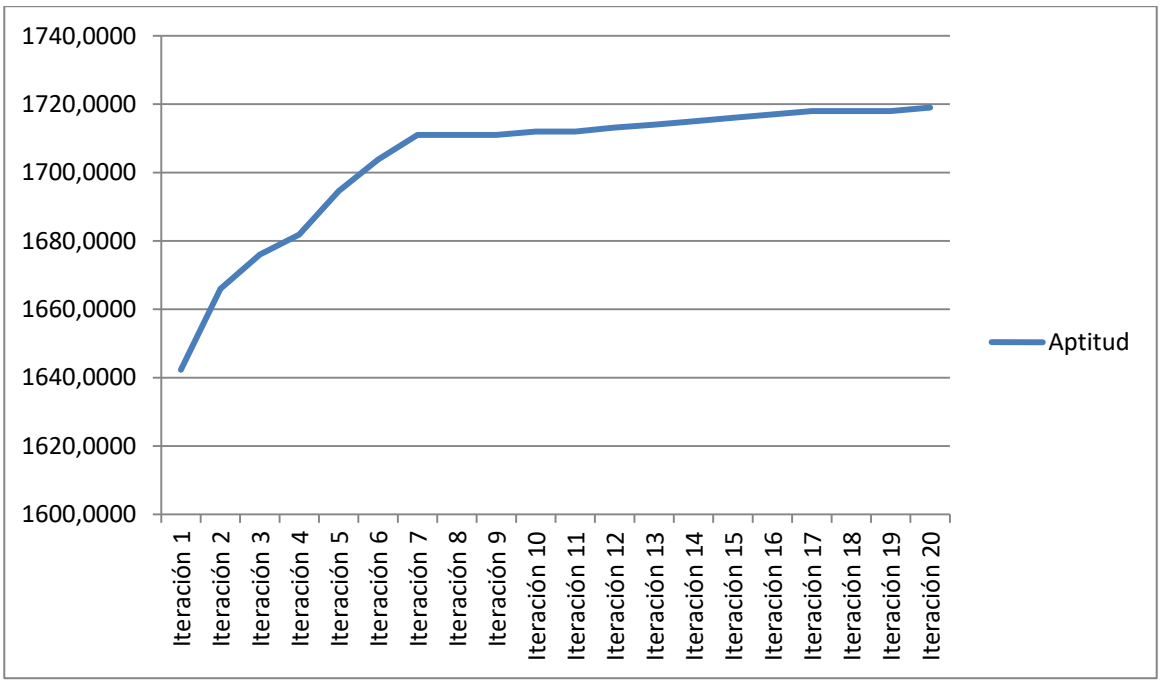

Figura 22. Gráfico de la evolución de la generación para una población de 32 viajeros.

El eje de las abscisas de la gráfica representa la iteración del algoritmo, mientras que el eje de las ordenadas representa el valor de la función aptitud obtenido.

Como se aprecia en la gráfica, al existir mas viajeros, aumenta el número de combinaciones posibles y eso tiene como repercusión que al algoritmo le cuesta encontrar la máxima solución válida posible. De hecho, según el gráfico, problablemente si existieran más de 20 iteraciones mejoraría aún más la solución obtenida. Este caso se asemeja más a la realidad en la que existe un gran número de viajeros que quieren viajar, y menos viajes disponibles que viajeros (ver Figura 22).

En la gráfica se observa que en cada generación(iteración del algoritmo) se mejora la máxima solución válida encontrada hasta el momento. A continuación se muestra el resultado del cromosoma cuya combinación de genes tiene la mayor función aptitud posible cumpliendo las restricciones del problema. Esta solución es la solución mostrada en la iteración 20 del algoritmo.

```
Mejor individuo total obtenido :
[2,2,2,2,2,1,1,4,3,4,2,2,3,1,1,4,3,2,3,2,3,4,1,4,2,2,3,4,2,2,2,4]
= 22222114342231143232341422342224

Mejor valor aptitud total obtenido: 1719.0
```

Dicho cromosoma es la mejor solución válida del problema para una población de 32 cromosomas, ya que no hay ninguna otra combinación de genes distinta a la actual que mejore el valor aptitud : **1719.0**.

Además, el algoritmo es capaz de devolvernos también el mejor individuo, que en este caso es el cromosoma cuya combinación de genes es la siguiente: **22222114342231143232341422342224**. A diferencia de la configuración con 16 viajeros, la longitud del cromosoma es de 32 números, siendo cada número el identificador del viaje que se le ha asignado.

6. Conclusiones

Respecto a la combinación de operadores genéticos del problema, se ha demostrado que usando una combinación de mutación y cruce, se obtiene unos resultados mejores y más rapidamente que usando otras configuraciones de operadores genéticos.

Sin embargo, usar únicamente la mutación como operador genético, es una mala opción, ya que los resultados obtenidos no son proporcionales al incremento de la población de cromosomas como si que ocurre con otros operadores.

Tras analizar el algoritmo con distinto número de viajeros, se ha llegado a la conclusión de que en poblaciones de pocos viajeros, el algoritmo muestra una solución válida en muy pocas generaciones (iteraciones). Sin embargo, a medida que se aumentan los viajeros, aumentan las posibles combinaciones de genes y el algoritmo necesita más generaciones para obterner una buena solución válida. No obstante, se ha demostrado que la máxima solución válida encontrada hasta el momento va aumentando a medida que aumentan las generaciones (iteraciones) del algoritmo.

.Por otra parte, se demostrado que si se aumenta la población inicial de cromosomas, el tiempo que tarda el algoritmo se incrementa. En conclusión, tras comparar las distintas configuraciones, observamos que la mejor configuración en relación población/operadores genéticos es la formada por la combinación de operadores genéticos de mutación y cruce y una población inicial de 600 cromosomas.

En general, es cierto que otras técnicas pueden garantizar una solución óptima al problema de la formación de grupos y la técnica de los algoritos genéticos no te la garantiza, pero si garantiza soluciones válida que satisfacen las restricciones del problema en un intervalo de tiempo muy reducido. Es por eso, y por su rápida respuesta a este tipo de problemas que el algoritmo genético es una opción totalmente viable para la formación de grupos para viajar en distintos viajes asignados.

Por ello, podemos garantizar que la utilización del algoritmo genético es adecuada para la generación automática grupos para viajar en distintos viajes asignados y podría perfectamente ser extrapolable a otros entornos.

6.1. Trabajos futuros

Este algoritmo podría ser la base para la realización de una aplicación informática que permita a los usuarios poder viajar de la forma que mas satisfaga sus necesidades. En este caso, este algoritmo permetiría al usuario satisfacer su presupuesto disponible, su lugar de salida y su preferencia por el destino del viaje, así como su relación con otras personas que relicen el mismo viaje.

Este algoritmo también podría utilizarse para sustituir otros algoritmos mas costosos computacionalmente que muestren unos resultados similares.

7. Bibliografía

[1] Lisset Alexandra Neyra Romero, Trabajo Final de JGAP https://es.scribd.com/document/59654444/Trabajo-Final-de-JGAP, 08-Jul-2011, [Último acceso: 22-Nov-2016]

[2] Jose Carlos López, Introducción a los algoritmos genéticos: como implementar un algoritmo genético en JAVA, https://www.adictosaltrabajo.com/tutoriales/jgap/#06 ,07-Oct-2010, [Último acceso: 22-Nov-2016]

[3] API JGAP, Package org.jgap, http://anji.sourceforge.net/javadoc/org/jgap/package-summary.html, [Último acceso: 24-Nov-2016]

[4] Anónimo,BlaBlaCar, https://es.wikipedia.org/wiki/BlaBlaCar, 23-Jun-2016, [Último acceso: 20-Nov-2016]

[5] Anónimo, Categoría:Compartición de cocheshttps://es.wikipedia.org/wiki/Categor%C3%ADa:Compartici%C3%B3n_de_coches, 14-Mar-2014, [Último acceso: 21-Nov-2016]

[6] API JGAP, Project: jgap_3.6.2_full, http://www.programcreek.com/java-api-examples/index.php?source_dir=jgap_3.6.2_full-master/epes-sgm-restful/src/main/java/org/epes/sgm/dmm/Scheduler.java, [Último acceso: 25-Nov-2016]

[7] API JGAP, org.jgap.impl.MutationOperator, http://grepcode.com/file/repo1.maven.org/maven2/cn.apiclub.third/jgap/3.6.2/org/jgap/impl/MutationOperator.java , [Último acceso: 19-Nov-2016]

[8] API JGAP, org.jgap.impl.CrossoverOperator, http://grepcode.com/file/repo1.maven.org/maven2/net.sf.jgap/jgap/3.4.4/org/jgap/impl/CrossoverOperator.java#CrossoverOperator.%3Cinit%3E%28org.jgap.Configuration%2Cdouble%29, [Último acceso: 18-Nov-2016]

[9] Tema 4.2.- Metaheurísticas Poblacionales (evolutivas): Algoritmos Genético(Dpto. Sistemas Informáticos y Computación - Universidad Politécnica de Valencia)

[10] Hernandez, J., Costa, A., Del Val, E., Alberola, J. M., Novais, P., and Julian, V., Using Genetic Algorithms for Group Activities in Elderly Communities, AT 2016 pp. In Press. (2016)

[11] Alberola, J. M., Del Val, E., Sanchez-Anguix, V., Palomares, A., & Teruel, M. D. An artificial intelligence tool for heterogeneous team formation in the classroom. Knowledge-Based Systems, 101, 1-14. (2016)

[12] Anónimo, car2go, https://es.wikipedia.org/wiki/Car2go, 22-Jun-2016, [Último acceso: 25-Nov-2016]

[13] Anónimo, Autolib', https://es.wikipedia.org/wiki/Autolib', 24- Ago-2015, [Último acceso: 18-Nov-2016]

Agradecer la estrecha colaboración en información (datos, gráficas y documentación) y ayuda a:

- Vicente Julián Inglada (tutor del proyecto)
- Elena del Val (codirectora)
- Juan M. Alberola (codirector)