Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación de Algoritmos Genéticos para la optimización de problemas combinatorios

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Juan Andrés Ambuludi Olmedo

Tutor: Federico Barber Sanchis

Curso 2017 - 2018

Resumen

El problema tratado en el presente trabajo consiste en la minimización del tiempo y del consumo energético utilizado por un vehículo eléctrico durante un recorrido. Este problema se encuentra motivado por la reducida autonomía de los vehículos eléctricos y la búsqueda de ahorro por parte de los conductores, que se beneficiarían de un método de conducción eficiente que les permitiese recorrer mayores distancias sin tener que detenerse a recargar el vehículo. Las principales dificultades son dos: Por una parte, la búsqueda de una solución óptima requiere un tiempo de cómputo elevado debido a que el número de soluciones al problema aumenta exponencialmente con la cantidad de diferentes tramos que conforman una carretera; Adicionalmente, los objetivos de minimizar tiempo y consumo resultan conflictivos pues aumentar la velocidad acorta la duración del recorrido a cambio de aumentar el consumo, pero reducirla tiene el efecto contrario. La forma de afrontar estos problemas pasa por la utilización de algoritmos genéticos, dentro de los cuales se encuentra NSGA II: Un algoritmo bioinspirado que trata problemas multiobjetivo realizando la búsqueda inteligente de una solución compromiso. La implementación del NSGA II se realiza tras el desarrollo de un algoritmo genético simple (SGA) ya que constan de elementos similares. El tener un algoritmo simple y uno especializado motiva la realización de comparaciones, sin embargo, los SGA tratan un solo objetivo por lo que es adaptado para tener en cuenta tiempo y consumo. La comparación se lleva a cabo en 20 carreteras diferentes en las cuales NSGA II consigue mejores valores de tiempo y consumo que SGA permitiendo concluir que su uso es recomendable cuando se busca un mayor beneficio.

Palabras clave: Vehículo eléctrico, algoritmo genético, optimización multiobjetivo, NSGA II.

Abstract

The problem addressed in this paper is to minimize the time and energy consumption used by an electric vehicle during a journey. This problem is caused by the reduced autonomy of electric vehicles and the search for savings by drivers, who would benefit from an efficient driving method that would allow them to travel longer distances without having to stop to recharge the vehicle. The main difficulties are two: On one hand, the search for an optimal solution requires a high computational time because the number of solutions to the problem increases exponentially with the number of distinct sections that make up a road; On the other hand, minimizing time and consumption is difficult since increasing the speed shortens the duration of the journey in exchange for increasing consumption, but reducing it has the opposite effect. The way to face these problems is using genetic algorithms, including NSGA II: A bio-inspired algorithm that addresses multi-objective problems by intelligently searching for a compromise solution. The implementation of NSGA II is done after the development of a simple genetic algorithm (SGA) as they have similar elements. Having a simple and specialized algorithm motivates comparisons, however, SGA treats only one objective, so it must be adapted to consider time and consumption. The comparison is carried out on 20 different roads on which NSGA II achieves better time and consumption values than SGA, allowing us to conclude that its use is recommended when looking for greater benefit.

Keywords : Electric vehicle, genetic algorithm, muti-objective optimization, NSGA II.

Tabla de contenidos

1. Introducción 7
   1.1. Problema 7
   1.2. Objetivos 8
   1.3. Metodología 9
   1.4. Estructura 9
2. Estado del arte 11
3. Problema 13
   3.1. Modelos 14
      3.1.1. Modelización del vehículo 14
      3.1.2. Modelo de la carretera 16
   3.2. Especificación 16
      3.2.1. Especificación del trayecto 17
      3.2.2. Especificación del vehículo y las fuerzas sobre él 18
   3.3. Objetivos 21
   3.4. Resumen 21
4. Diseño y desarrollo 23
   4.1. Métodos existentes 23
   4.2. Metaheurísticas 28
      4.2.1. Algoritmos genéticos. 30
      4.2.2. Algoritmos genéticos multiobjetivo 32
   4.3. Diseño de un GA simple 34
      4.3.1. Diseño de los individuos 34
      4.3.2. Operadores genéticos 37
      4.3.3. Tratamiento de múltiples objetivos 40
   4.4. Diseño de un AG multiobjetivo: NSGA II 41
      4.4.1. Procedimientos del algoritmo NSGA II 41
      4.4.2. Diseño del individuo 44
      4.4.3. Operadores genéticos 44
5. Implementación 47
   5.1. Lenguaje de programación 47

5.2. Entorno de desarrollo 48
5.3. Detalles de implementación 49
5.3.1. Implementación del algoritmo genético simple 49
5.3.2. Implementación del algoritmo NSGA II 56
6. Experimentación: Evaluación y pruebas 59
6.1. Realización del experimento 59
6.2. Datos del escenario de aplicación 60
6.2.1. Configuración del vehículo 60
6.2.2. Configuración de la carretera 61
6.2.3. Configuración de los algoritmos SGA y NSGA II 63
6.3. Casos de prueba 63
6.3.1. Pruebas previas: Tiempo y consumo en SGA. 63
6.3.2. Pruebas SGA 65
6.3.3. Pruebas NSGA II 68
6.3.4. Conclusiones sobre los experimentos 71
7. Conclusiones 73
7.1. Resumen del trabajo realizado. Interés, resultados 73
7.2. Posibles ampliaciones 74
7.3. Relación con estudios cursados 75
8. Bibliografía 77

1. Introducción

Actualmente, el uso de vehículos está ampliamente extendido, son utilizados a diario, los fines de semana y también para vacaciones debido a la comodidad que suponen y la posibilidad de acceder a lugares a los que sería difícil llegar con transporte público. A pesar de sus ventajas, no se encuentran exentos de inconvenientes como la contaminación producida y el precio del combustible o la baja autonomía en el caso de los vehículos eléctricos.

El ahorro del combustible en un vehículo de combustión se puede conseguir conduciendo de forma eficiente [1], como por ejemplo manteniendo una velocidad constante, moderando la velocidad, usando el freno motor para desacelerar en lugar de pulsar el pedal del freno y aprovechando las pendientes descendentes. Conducir un vehículo eléctrico siguiendo las mismas reglas es especialmente importante pues permite conducir durante más tiempo sin tener que recargar la batería, compensando así su baja autonomía.

Una conducción eficiente no solo genera ahorro energético, sino también en el precio de la póliza de seguro ya que existen aseguradoras de coches que ofrecen seguros denominados *Pay as you drive*[2] (Paga como conduces). Estos seguros se sirven de dispositivos que analizan la forma de conducir para identificar "buenos conductores", los cuales se benefician de un precio más bajo. Estos dispositivos examinan factores como el tiempo de conducción sin descanso, los kilómetros recorridos, los tipos de carreteras por las que circula o la velocidad.

Sin embargo, conducir eficientemente no significa simplemente conducir a bajas velocidades ya que el hacerlo aumentaría la duración del viaje y obligaría al conductor a hacer un mayor número de paradas, incrementando aún más el tiempo que dura el trayecto. Una buena conducción dependerá de la capacidad del conductor para hallar una solución compromiso que minimice la duración del viaje y permita conseguir el ahorro en el consumo.

1.1. Problema

La situación descrita al principio de este capítulo pone de manifiesto la necesidad de encontrar formas de conducir que faciliten a los conductores el saber adaptarse al terreno por el que conducen. La búsqueda de estos modos de conducción puede plantearse como un problema de optimización en el que el objetivo final consiste en reducir simultáneamente el tiempo y el consumo.

[1] http://revista.dgt.es/es/educacion-formacion/conducir-mejor/2017/0213-Conduccion-eficiente-diez-claves-para-consumir-menos.shtml#.Wz-rq7jtZEZ
[2] http://www.seguros.es/blog/seguro-de-coche-pay-as-you-drive

Una carretera puede verse como un conjunto de tramos en el que cada tramo tiene una velocidad "adecuada" que dependerá de las características del tramo como por ejemplo el estado del suelo o la inclinación. La velocidad adecuada u óptima será aquella que permita superar el tramo en el menor tiempo y con el menor gasto energético.

La velocidad a la que se desplaza un vehículo depende de la presión que el conductor ejerce sobre el pedal por lo que para obtener la velocidad adecuada es necesario encontrar previamente la presión sobre el pedal que la provoca. Una búsqueda entre los niveles de presión posibles sería viable en un tramo, sin embargo, las combinaciones de niveles aumentan exponencialmente con la cantidad de tramos distintos.

La situación se complica si se advierte que los objetivos son conflictivos: Al aumentar la velocidad para reducir el tiempo, el consumo aumenta. Si disminuye la velocidad el consumo disminuye, pero el tiempo aumenta.

Teniendo en cuenta el gran espacio de búsqueda y los objetivos del problema, se recurre a métodos que buscan la solución de manera inteligente: Los algoritmos genéticos, cuya capacidad para buscar de forma inteligente buenas soluciones permite eludir (aunque no eliminar) el problema del crecimiento exponencial a cambio de no garantizar una solución óptima. Dentro de los algoritmos genéticos se encuentra el NSGA II, un algoritmo capaz de hallar soluciones compromiso en problemas con varios objetivos como es el caso de este problema.

1.2. Objetivos

El principal objetivo del proyecto es conseguir una forma de minimizar el tiempo y el consumo de un vehículo durante su recorrido por la carretera. No obstante, un resultado correcto no es suficiente, sino que ha de obtenerse empleando la menor cantidad de recursos posible (tiempo de ejecución, almacenamiento, …). Del mismo modo, es deseable que la solución pueda ser reutilizada. Un ejemplo de reutilización sería que un mismo vehículo funcionase correctamente en diferentes carreteras para evitar recalcular la solución al cambiar de autopista.

Como objetivo transversal se encuentra el análisis de diferentes formas de resolución del problema. Concretamente, se enfrenta un algoritmo genético simple, que trata los objetivos del problema como uno solo, contra un algoritmo genético especializado en resolver problemas con más de un objetivo: El NSGA II. Además de esto, se ponen a prueba diferentes tipos de carretera para analizar el algoritmo que funciona mejor en la mayoría de las situaciones.

1.3. Metodología

La metodología utilizada en el proyecto es la siguiente:

En primer lugar, se definen los modelos que se usarán durante todo el proyecto, concretamente se define un modelo de vehículo y de carretera que facilitan la experimentación con el problema planteado.

Una vez conseguidos los modelos, se procede a la búsqueda de un método de resolución que permita cumplir con el requerimiento de consumir pocos recursos, para esto, se hace un análisis del coste temporal del problema y posteriormente se revisan las algunas de las técnicas existentes para tratar problemas similares, destacando entre ellas el uso de algoritmos genéticos.

Tras escoger una forma para resolverlo, se procede a su implementación en el lenguaje de programación Python. Los algoritmos SGA y NSGA II son puestos a prueba en una en 20 diferentes tipos de carretera para comprobar su comportamiento en múltiples casos. Tras la experimentación, se realiza una crítica de los resultados en la que se analizan los datos obtenidos

Por último, se hacen comentarios generales sobre el proyecto como el interés general de su realización y la justificación de los métodos escogidos, además de posibles ampliaciones.

1.4. Estructura

El contenido del presente trabajo se organiza como sigue:

Este primer capítulo contiene información básica sobre el problema, los objetivos que se persiguen y la metodología empleada para lograrlos.

La sección 2 contiene el estado del arte. En ella se relata el contexto actual que rodea al problema y se mencionan las técnicas relacionadas con este trabajo.

En la sección 3 se describe el problema en detalle y se modelan los elementos que forman que forman parte de él: Vehículo y carretera. Dicho modelo incorpora la relación de las fuerzas que actúan sobre un vehículo en movimiento, los parámetros necesarios para definir un automóvil, los parámetros de la carretera, etc. Al final de la sección se realiza una síntesis de lo visto en ella.

En el capítulo 4 se plantea la forma de resolver el problema, para ello, se hace un breve recorrido sobre las opciones disponibles para luego especificar el método escogido: Los algoritmos genéticos. Concretamente se elige un algoritmo genético simple y uno multiobjetivo. Dentro de este apartado se describe en qué consisten dichos algoritmos, los pasos empleados para resolver problemas con ellos y los elementos que los conforman (operadores de selección, operadores de cruce, …).

En el apartado 5 se expone la implementación de los algoritmos elegidos en el capítulo 4 en un lenguaje de programación concreto. En el punto 5.1. se justifica la elección del lenguaje de programación *Python* y el entorno *PyCharm* como medio para la implementación, también se explican los procedimientos y funciones utilizadas. El código completo se encuentra en un repositorio privado de BitBucket.

En el capítulo 6 se ponen a prueba las implementaciones desarrolladas. Junto a las pruebas se comentan y justifican los resultados obtenidos. Las pruebas se centran en el análisis del comportamiento de SGA y NSGA II en 20 diferentes carreteras con tres operadores de cruce distintos: Cruce de dos puntos, multipunto y aleatorio.

El apartado 7 se divide en tres puntos: En 7.1. se resume el problema junto a la motivación e interés de su realización y se comentan los resultados obtenidos en el capítulo de experimentación; En el punto 7.2. se explica cómo se relaciona el proyecto con los estudios cursados y su utilidad para completarlo; La sección 7.3. corresponde a los trabajos futuros. En ella se señalan cambios se podrían realizarse para ampliar el proyecto. También se mencionan algunas pruebas que sería interesante realizar para comprobar aún más las capacidades del desarrollo.

Por último, el apartado 8 contiene las referencias bibliográficas de las cuales se ha extraído la información necesaria para llevar a cabo el trabajo. El formato escogido es el estándar MLA 8ª edición.

2. Estado del arte

Los problemas de optimización son objeto de estudio continúo dadas sus diversas utilidades: Cortes de materias primas en la industria, planificación de horarios para lugares de trabajo, cantidades de los ingredientes en la industria alimentaria, etc. Algunos de estos problemas suelen ser tratados con el método simplex desarrollado por George Bernard Dantzig. Otro método para resolver problemas de optimización son los algoritmos evolutivos. En el libro "Algoritmos Evolutivos: Un enfoque práctico" de Lourdes Araujo y Carlos Cervigón se hallan varios ejemplos de optimización junto a guías de cómo resolverlos, también se incluye un repaso de las nuevas tendencias como son la inteligencia colectiva y algoritmos de colonias de hormigas.

Dentro del ámbito de la optimización se encuentran problemas donde no existe un solo objetivo sino varios, un ejemplo de esto representa la fabricación de químicos, en la cual hay que controlar el coste de la elaboración y las emisiones generadas. Debido a su capacidad para tratar con múltiples soluciones los algoritmos genéticos han ganado popularidad a la hora de resolver este tipo de problemas.

En el libro de Carlos A, Coello Coello "*Evolutionary Algorithms for Solving Multi-Objective Problems"* se realiza una extensa explicación sobre los algoritmos evolutivos y los métodos derivados de estos. Algunos ejemplos que se pueden encontrar son: Multiple-Objetive Genetic Algorithm (MOGA), Vector Evaluated Genetic Algorithm (VEGA), Strength Pareto Evolutionary Algorithm (SPEA), Nondominated sorting genetic algorithm II (NSGA II), entre otros. Además, se incluye un repaso de técnicas recientes para mejorar las capacidades de los algoritmos evolutivos como la coevolución y la simbiosis.

El problema del proyecto entra en el grupo de los problemas de optimización multiobjetivo. Los objetivos corresponden a la minimización del tiempo y el consumo en automóviles. Existen trabajos relacionados con la minimización de la velocidad (y en consecuencia el tiempo) y el consumo de combustible en aviones. Un ejemplo de esto es el documento "Cruise Fuel Reduction Potential from Altitude and Speed Optimization in Global Airline Operations" de Luke L. Jensen, Henry Tran, y R. John Hansman.

El plantear la optimización sobre un vehículo requiere disponer de un modelo que tenga en cuenta su dinámica. Afortunadamente, existen multitud de modelos debido a su uso en videojuegos de conducción[3] y en simuladores. Un ejemplo de modelado se puede encontrar en el trabajo de Helmut Hlavacs "A 2D Car Physics Model based on Ackermann Steering".

El presente trabajo se beneficia de modelos de vehículos existentes y de los algoritmos genéticos para resolver el problema que supone reducir el combustible empleado en un trayecto junto al tiempo empleado utilizado.

[3] http://www.asawicki.info/Mirror/Car Physics for Games/Car Physics for Games.html

3. Problema

Como se ha adelantado en la sección de introducción, el problema a resolver consiste en mejorar el tiempo empleado en el recorrido en vehículo a la vez que se procura reducir el consumo. Una sencilla solución para llegar al destino elegido en el menor tiempo posible sería pisar el acelerador desde el primer momento con la única preocupación de no sobrepasar los límites de velocidad permitidos. Respecto al segundo objetivo, el consumo, se produciría un ahorro conduciendo a una velocidad moderada y aprovechando los tramos descendientes. La complejidad aparece cuando se trata de conciliar ambos objetivos pues una mayor velocidad y en consecuencia un menor tiempo, implica aumentar el consumo de combustible. Si se trata de reducir el consumo se aumenta inevitablemente la duración del viaje.

Del mismo modo, es necesario tener en cuenta el medio por el que circula el conductor ya que las fuerzas que actúan sobre el automóvil dependerán de factores como la inclinación de la carretera, si esta se encuentra pavimentada o no e incluso la densidad del aire puesto que ofrece resistencia a su avance. La combinación de un modelo que tenga en cuenta las fuerzas a las que se ve sometido un vehículo junto a un modelo de carretera permite obtener el comportamiento dinámico del vehículo.

La situación planteada muestra un problema de optimización multiobjetivo en la que las metas entran en conflicto y en el que hay que tener en cuenta múltiples factores por lo que es de esperar que el coste computacional sea elevado lo que representa otro punto a tener en cuenta.

Aunque la forma concreta de la solución y de las funciones del problema se desarrollan en el capítulo 4, se puede dar una idea aproximada del funcionamiento general: El problema consta de dos funciones f_1 y f_2 que reciben como argumentos un comportamiento x y una carretera r. El comportamiento x sobre r es comprobado por f_1 y f_2 que retornan el valor del tiempo y el consumo generado en r respectivamente.

$$f_1(x, r) = tiempo$$

$$f_2(x, r) = consumo$$

El objetivo es entonces minimizar ambas funciones simultáneamente:

$$\min\big(f_1(x, r),\ f_2(x, r)\big) = (min.tiempo, min.consumo)$$

3.1. Modelos

Antes de entrar en la resolución del problema, es necesario entender mejor los componentes que participan en él y diseñar modelos que recojan sus características a fin de facilitar la resolución del problema. En este apartado se explican dichos modelos junto a su contribución en la obtención de la solución.

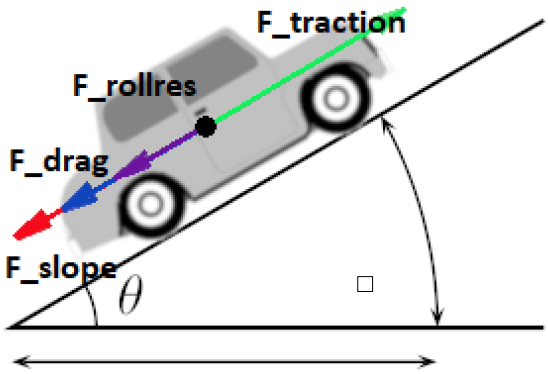

Ilustración 1. Fuerzas que actúan sobre un vehículo.

3.1.1. Modelización del vehículo

En los últimos años, los vehículos eléctricos (EV) han ido ganando una mejor posición en el mercado debido a la reducción de precios cercanos a los de los automóviles tradicionales y a mejoras en las baterías empleadas, que permiten una mayor autonomía haciéndolos más aptos para largos trayectos. La mayor utilización de estos medios de transporte hace necesario desarrollar perfiles de conducción específicos para que sus conductores hagan un uso más eficiente de ellos.

Algunas de las características internas de los EV son las siguientes (Brain 1):

- El motor de gasolina se sustituye por uno eléctrico.
- El motor eléctrico recibe su energía de un controlador.
- El controlador recibe su energía de baterías recargables.
- Los EV carecen de marchas, permitiendo una conducción más fácil.

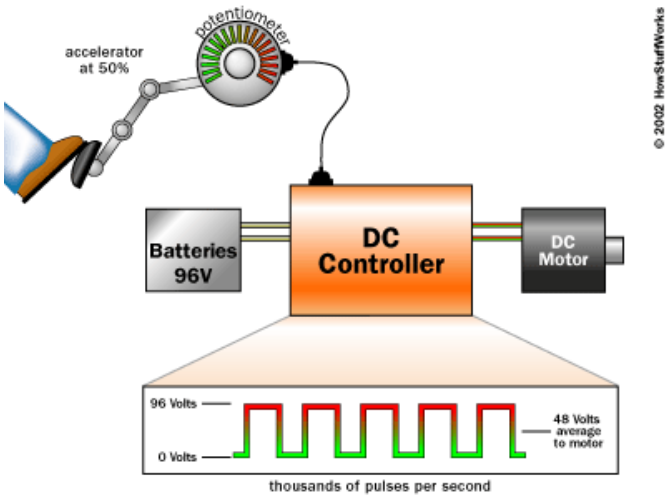

Ilustración 2. Presión del pedal y efecto en el motor (HowStuffWorks).

La marcha del automóvil empieza con la presión del acelerador. Esta presión sirve para regular el valor de un potenciómetro, que indica al controlador cuanta potencia ha de enviar al motor. Si el conductor presiona totalmente el acelerador, el controlador envía los 96V de las baterías al motor, si no se pulsa se transfieren 0V. Para valores intermedios el voltaje se divide en segmentos de la misma longitud (Brain 2).

La posición del pedal puede verse como un valor en el rango 0-100, siendo 0 la ausencia de presión y 100 una la presión máxima. Los valores de voltaje se obtienen según la expresión: $(Máximo\ voltaje) * (presión\ sobre\ el\ pedal\ /100)$.

Los EV tienen curvas velocidad – par motor (curvas sólidas) y curvas velocidad – Potencia (curvas punteadas) como las siguientes:

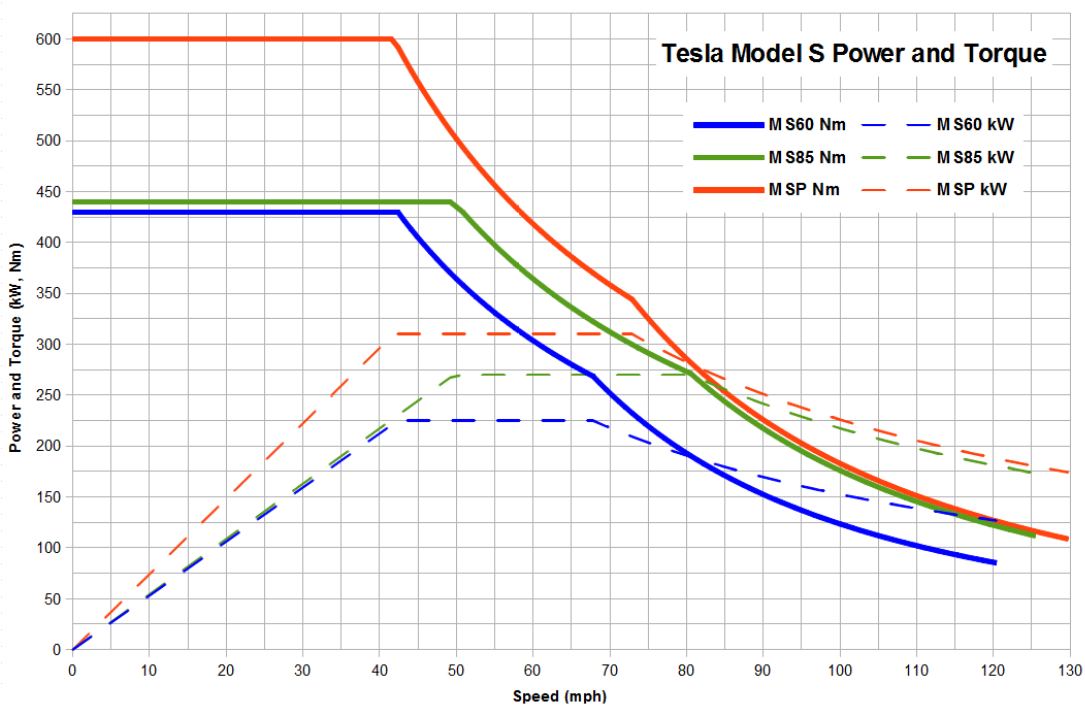

Ilustración 3. Curvas velocidad-Potencia (líneas punteadas) y velocidad-par motor (líneas sólidas).

Se puede observar en las líneas solidas que se dispone del par motor máximo desde el principio, permitiendo aceleraciones de 0-100 km/h en 2.28 segundos (Valdes-Dapena, párr. 1). En este caso concreto del *Tesla Model S M S60* el valor del par motor se mantiene constante hasta los 42.5 mph aproximadamente. A partir de ese instante el par desciende a medida que aumenta la velocidad.

La velocidad y aceleración (positiva o negativa) del vehículo en cada momento dependerá de la suma de fuerzas que actúan sobre él y de la masa del vehículo. Sobre el EV interactúan fuerzas a favor y en contra:
A favor: Fuerza de tracción, Pendiente positiva de la carretera.
En contra: Fuerzas de rozamiento y mecánicas, Fuerzas contra el avanza debidas al viento, Pendiente negativa de la carretera.

3.1.2. Modelo de la carretera

Una vez definido el modelo del vehículo, el siguiente paso consiste en describir el entorno en el que se mueve.

El medio consiste en una carretera bidimensional en la cual el automóvil puede desplazarse hacia arriba, hacia abajo y hacia adelante. Debido a la gran extensión de las carreteras, se realizan subdivisiones denominadas secciones o tramos. Cada uno de estos tramos tiene cuatro características: Inclinación, densidad del aire, longitud y una velocidad máxima permitida situada al final del tramo. Aunque pueda parecer que la densidad del aire no es una característica propia del terreno, un tramo ha de entenderse como el ambiente en el que se encuentra el EV y no solamente el suelo por el que circula.

Las carreteras pueden ser clasificadas en tres grupos en función de la inclinación de sus tramos: predominantemente positivas, predominantemente negativas y neutrales.

1. **Predominantemente positivas** si la mayoría de los tramos que las componen tienen inclinación mayor que 0.
2. **Predominantemente negativas** si la mayoría de los tramos que las componen tienen inclinación menor que 0.
3. **Neutrales** si la mayoría de los tramos que las componen tienen inclinación igual a 0.

3.2. Especificación

En este punto se concretan los modelos descritos en 3.1 definiendo los parámetros que son necesarios para crear prototipos a partir de ellos. En el caso del vehículo se describen también las fuerzas a las que se encuentra sometido durante su recorrido y las fórmulas necesarias para calcular el tiempo y el consumo ocupados en un tramo.

3.2.1. Especificación del trayecto

El viaje se plantea como una serie de tramos, cada uno con una inclinación, longitud, velocidad máxima y densidad del aire en el tramo.

3.2.1.1. Parámetros y variables de una carretera

Tramos: Conjunto de estructuras que contienen información sobre la inclinación, longitud, velocidad máxima y densidad del aire en un tramo. La unión de todos los tramos forma la carretera.

Mínima inclinación: Mínimo valor del parámetro inclinación en el conjunto de tramos.

Máxima inclinación: Máximo valor del parámetro inclinación en el conjunto de tramos.

Precisión: Indica las subdivisiones de la inclinación. Los valores de precisión disponibles son: 0.1, 0.2, 0.5, 1. Una precisión de 0.1 indica 1/0.1 = 10 subdivisiones por cada grado (%) de inclinación.

Parámetro	Unidad	Símbolo
Conjunto de tramos	–	–
Mínima inclinación	%	–
Máxima inclinación	%	–
Precisión	–	–

Tabla 1. Parámetros de una carretera: Unidades y símbolos

3.2.1.2. Parámetros de un tramo de carretera

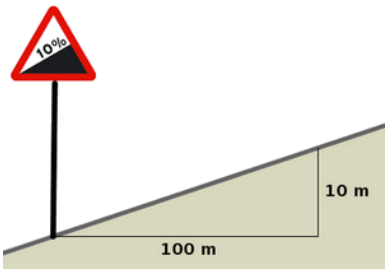

Ilustración 4. Inclinación de un tramo

Inclinación: es la tangente del ángulo que forma la carretera con la horizontal (López, párr. 3).

Densidad del aire: Relación entre la masa y el volumen del aire, siendo su valor 1.225 kg/m3 a la presión atmosférica normal y a 15 ºC, aunque disminuye con la humedad, la temperatura, y la altitud, y por tanto influye en la energía cinética del viento (Real Academia de Ingeniería, párr. 3).

Coeficiente de resistencia a la rodadura: La resistencia a la rodadura aparece cuando las ruedas giran sobre el terreno. Su valor depende de las condiciones de la carretera y del tipo de neumáticos. Unos neumáticos normales sobre asfalto tienen un valor de 0.02 (*Engineering ToolBox*, párr. 4).

Velocidad máxima permitida: Se encuentra delimitada mediante señales de tráfico.

Longitud del tramo: Duración en metros de cada una de las divisiones del trayecto.

A pesar de tener varios parámetros que pueden distinguir un tramo de otro, en el proyecto se tiene en cuenta solo la inclinación cuando se hace referencia al término "tramos diferentes".

Parámetro	Unidad	Símbolo
Inclinación	$\%$	$-$
Coeficiente de resistencia a la rodadura	$-$	fr
Densidad del aire	kg/m^3	ρ
Velocidad máxima permitida	m/s	v_{max}
Longitud del tramo	m	$-$

Tabla 2. Parámetros de un tramo: Unidades y símbolos.

3.2.2. Especificación del vehículo y las fuerzas sobre él

3.2.2.1. Parámetros y variables de un vehículo

Área de arrastre (m^2): El área de arrastre es un valor que se obtiene mediante el producto entre el coeficiente de arrastre y el área frontal (m^2) del vehículo. Esta variable se encuentra relacionada con la aerodinámica del vehículo.

Masa (kg): La masa del EV.

Batería: La potencia almacenada de la que disponen las baterías del EV.

Radio de la rueda (m): Radio de las ruedas. Se asume que las cuatro ruedas tienen el mismo nivel de inflado.

Transmisión: Es la relación de las velocidades de dos engranajes conectados. Esta relación se representa mediante $n_1{:}n_2$ donde por cada n_1 vueltas de entrada se genera n_2 de salida, generalmente $n_2 = 1$. El EV modelo dispone de una única marcha.

Proporción del diferencial: Relaciona la velocidad de salida de la transmisión con la velocidad de las ruedas.

Eficiencia de la transmisión: Es un factor que tiene en cuenta las pérdidas de energía. Se encuentra en el intervalo [0, 1]. Una eficiencia igual a uno indica la ausencia de pérdidas.

Aceleración máxima(m/s^2): La máxima aceleración del EV. Se puede obtener un valor de aceleración máxima a partir de las especificaciones mediante la fórmula:

$$a = \frac{\Delta V}{t}$$

Un objeto que alcanza los $100\ m/s$ en 2.28 segundos tiene una aceleración de:

$$\frac{(100 - 0)}{2.28} = 43.86\ m/s^2$$

Puntos velocidad-torque $(m/s, Nm)$: Un conjunto de tuplas (velocidad, par motor) que indican el par máximo a una determinada velocidad.

Puntos velocidad-consumo $(m/s, KW)$: Un conjunto de tuplas (velocidad, consumo) que indican el consumo máximo a una determinada velocidad.

Parámetro	Unidad	Símbolo
Área de arrastre	m^2	A
Masa	kg	w
Batería:	kW/h	b
Radio de la rueda	m	r
Caja de cambios	$-$	$gear$
Proporción del diferencial	$-$	dif
Eficiencia de la transmisión	$-$	tef
Aceleración máxima	m/s^2	a_{max}
Puntos velocidad-torque	$m/s, Nm$	$-$
Puntos velocidad-consumo	$m/s, KW$	$-$

Tabla 3. Parámetros de un vehículo: Unidades y símbolos.

3.2.2.2. Dinámica de un vehículo

La evolución de un vehículo a lo largo del tiempo se describe a través de las a las que se ve sometido en cada momento.

Resistencia a la Rodadura (Fernández 25): Presente en las ruedas al girar sobre una superficie, se calcula como:

$$F_{roll}\ (N) = fr * w * min(1, velocidad)$$

El término $min(1, velocidad)$ se incluye para evitar que esta fuerza sea cero cuando la velocidad también lo sea.

Fuerza pendiente (Fernández 25): Es la fuerza generada cuando se conduce un automóvil en una pendiente. En terreno llano equivale a cero.

$$F_{slope}\ (N) = w * seno\ (inclinación)$$

Resistencia aerodinámica (Fernández 25): La resistencia aerodinámica es una fuerza que se opone al avance del vehículo. Aumenta al aumentar la fuerza con la que avanza el automóvil.

$$F_{drag}\ (N) = 0.5 * \rho * A * velocidad^2$$

Esta fuerza aumenta proporcionalmente con el cuadrado de la velocidad por lo que cuanta más velocidad, más resistencia y un mayor consumo energético.

Fuerza tracción: La fuerza tracción corresponde a la fuerza con la que avanza el vehículo.

$$F_{traction}\,(N) = par\ en\ las\ ruedas/r$$

El par motor máximo se obtiene observando la gráfica de velocidad-par motor. El par real en las ruedas se calcula multiplicando el par máximo por la presión sobre el pedal entre 0 y 1, la eficiencia de la transmisión, la proporción del diferencial y la proporción de la marcha.

$$par\ en\ las\ ruedas = par\ máximo * gear * dif * tef * pedal$$

Fuerza total: Es el resultado del sumatorio de las fuerzas sobre el objeto.

$$F_{total}\,(N) = Ftraction - Froll - Fdrag \pm Fslope$$

El signo de la fuerza pendiente será positivo si es un ascenso y negativo en el caso de los descensos.

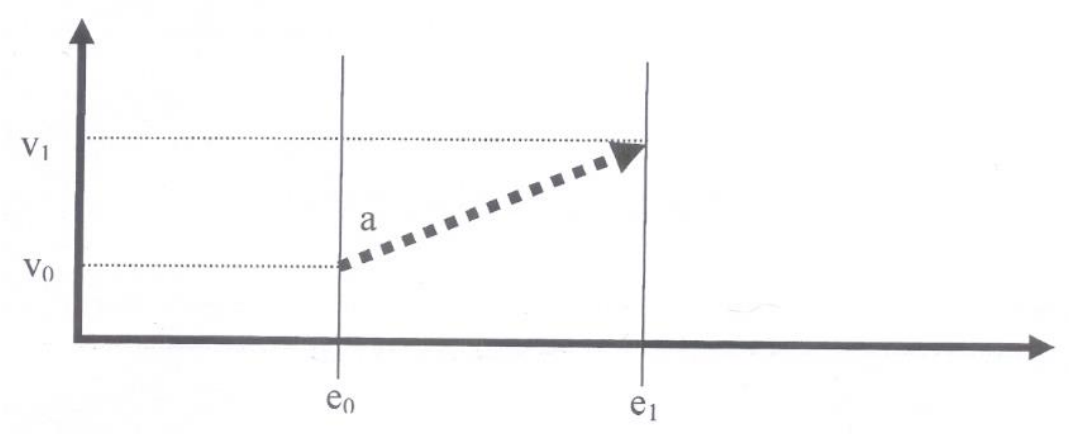

Ilustración 5. Relación entre velocidad, aceleración y espacio.

Aceleración: La aceleración se calcula con la fórmula de la segunda ley de movimiento de Newton. Si el resultado supera la aceleración máxima se reajusta.

$$a\ (m/s^2) = \min(F_{total}/w, a_{\max})$$

Tiempo en recorrer el tramo: se obtiene a partir de la expresión:

$$longitud\ del\ tramo\ (m) = e_1 - e_0 = velocidad_{inicial} * tiempo + 0.5 * a * tiempo^2$$

Velocidad al final del tramo: Depende de la velocidad inicial del vehículo y de la aceleración. Teniendo en cuenta el tiempo anterior:

$$Velocidad_{final}(m/s) = Velocidad_{inicial} + a * tiempo$$

Si se supera la velocidad máxima se reajustan la velocidad y el tiempo:

$$velocidad_{final} = \min(velocidad_{final}, v_{max})$$

$$tiempo = longitud\ del\ tramo\ /\ v_{\max}$$

Consumo: Para calcular el consumo se observa la gráfica para una velocidad determinada. El consumo en un tramo viene dado por el tiempo empleado en recorrerlo y el nivel de presión del pedal entre 0 y 1.

$$consumo = consumo\ (gráfica) * pedal * tiempo$$

3.3. Objetivos

Una vez establecidos los participantes del problema, el siguiente paso es decidir cómo usarlos para obtener una solución que satisfaga la motivación planteada al principio de este capítulo. La solución dependerá de la forma en la que un conductor guíe el vehículo. Su conducción se realizará basándose en lo que se denomina perfil de conducción. El principal objetivo pues, es obtener una guía que indique como ha de presionar el pedal un conductor para llegar al destino en el menor tiempo posible, a la vez que ahora la energía de la batería.

Además de un resultado correcto, este debe conseguirse procurando ahorrar tiempo de ejecución y memoria por lo que debe elegirse un método de resolución que tenga en cuenta estos recursos. Aunque se explica con más profundidad en secciones posteriores, se puede adelantar que el método consistirá en la aplicación de los denominados algoritmos genéticos.

A estos objetivos, se suma la intención de conseguir que el método escogido funcione en diversos casos, para ello se realizarán numerosas pruebas modificando algunas de sus características y variando los recorridos para un vehículo, de este modo se simulan diferentes situaciones a las que puede enfrentarse un conductor.

3.4. Resumen

En este apartado se ha hecho una descripción del problema que se trata en el proyecto y posteriormente del modelo de los elementos principales que participan en él: Carretera y vehículo eléctrico. Posteriormente, se han mencionado los parámetros y variables necesarios para instanciar los modelos cuando sea necesario realizar acciones con ellos.

Los objetivos que se persigue principalmente es la generación de una solución que minimice el tiempo y el consumo. Para ello, es necesario recurrir a métodos que sean capaces de optimizar simultáneamente ambas características.

4. Diseño y desarrollo

En este capítulo se describen los distintos enfoques que pueden seguirse para resolver el problema del capítulo 3. La sección 4.1. empieza con una categorización del problema atendiendo a su complejidad computacional y continúa con la descripción de algunos de los métodos de resolución capaces de manejar varios objetivos en un mismo problema como en el caso planteado. En la sección 4.2. se concreta el método escogido: Los algoritmos evolutivos, entre de los cuales se encuentran los algoritmos genéticos simples y el algoritmo NSGA II; En 4.3. y 4.4. se detallan los elementos necesarios para su implementación en un lenguaje de programación.

4.1. Métodos existentes

Los problemas de optimización, dentro de los cuales se encuentra el descrito en la sección anterior, tienen un coste de resolución de orden exponencial que hace inviable su resolución exacta. Veámoslo con un ejemplo:

Supongamos una carretera cuyos tramos solo pueden tener dos ángulos diferentes en total (por ejemplo -1 y 1 grados), por lo tanto, la solución será una lista de tres elementos donde cada uno será un nivel de presión del acelerador de entre los 101 posibles (entre 0 y 100 ambos inclusive). La ilustración 6 refleja la situación:

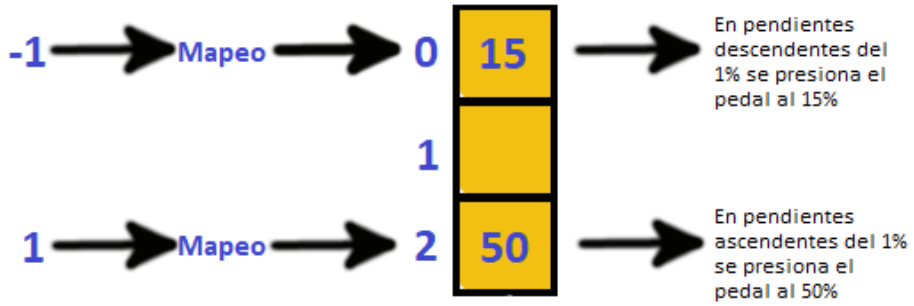

Ilustración 6. Niveles de presión en función de la pendiente

El conjunto X de soluciones factibles es:

$$X = \{(x, y, z): x, y, z \in \{0, \dots, 101\}\} =$$

$$\{(0,0,0), \dots, (0,0,101), (0,101,0), \dots, (0,101,101)\} \cup$$

$$\{(101,0,0), \dots, (101,0,101), (101,101,0), \dots, (101,101,101)\}.$$

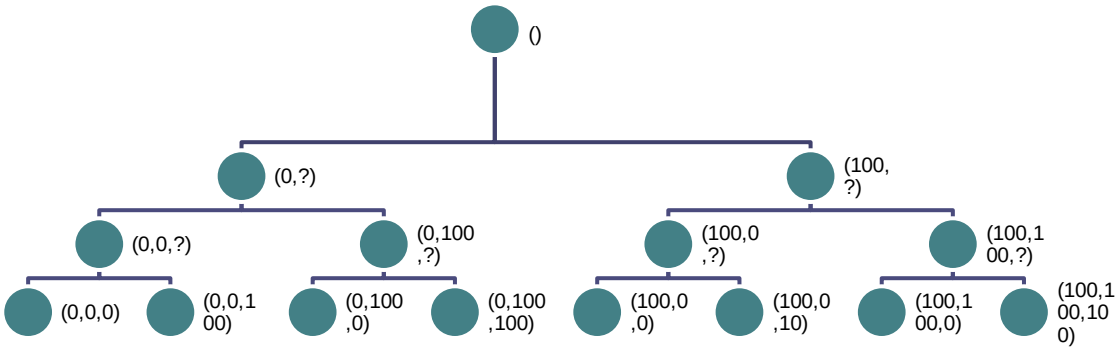

Ilustración 7. Conjunto de soluciones factibles

En este caso particular hay $101\,x\,101 = 10201$ soluciones para un problema relativamente pequeño y se podría resolver mediante fuerza bruta, esto es, recorriendo todas y cada una de las soluciones hasta encontrar la mejor. El problema viene dado cuando aumenta el número de inclinaciones tratadas y la cantidad de tramos ya que el cálculo de la función $f(x)$ tiene un coste lineal $O(n)$ con dicha cantidad.

El cálculo del número de soluciones se puede generalizar observando que la búsqueda de soluciones sigue una estructura en forma de árbol donde el factor de ramificación b corresponde a la cantidad de diferentes presiones posibles y el máximo nivel de profundidad a es el número de inclinaciones distintas. El total de número de nodos en el árbol se calcula como:

$$número\ de\ soluciones = número\ de\ nodos = b^a$$

Se puede llegar a la misma conclusión añadiendo b y a en la definición de X:

$$X = \{(x_1, \dots, x_a): x_1, \dots x_a \in \{0, \dots, b-1\}\}$$

X se reformula como:

$$X = (x_1, \dots, x_a), \qquad 0 \le x_i \le b-1$$

El número de elementos de X puede verse como un ejercicio de combinatoria en el que han de elegirse a elementos de entre b posibles con repetición:

$$|X| = |(x_1, \dots, x_a)| = b_0\ x \dots x\ b_a = b^a$$

Llegando a la misma conclusión.

Para una cantidad de 11 ángulos (por ejemplo -5, …, 0, … 5 grados) tendríamos tantas soluciones como estrellas en el universo ($10^{22}\ aprox$). Esto hace inviable su resolución por métodos de fuerza bruta ya que como se ha mostrado el coste es exponencial con el número de ángulos $O(101^a)$.

Existen métodos que harían una búsqueda exacta de la solución más eficiente como el uso de técnicas de *backtracking* o ramificación y poda que, si bien acotan el espacio de

búsqueda y consiguen reducir el tiempo de cálculo, el coste exponencial es ineludible en el caso peor.

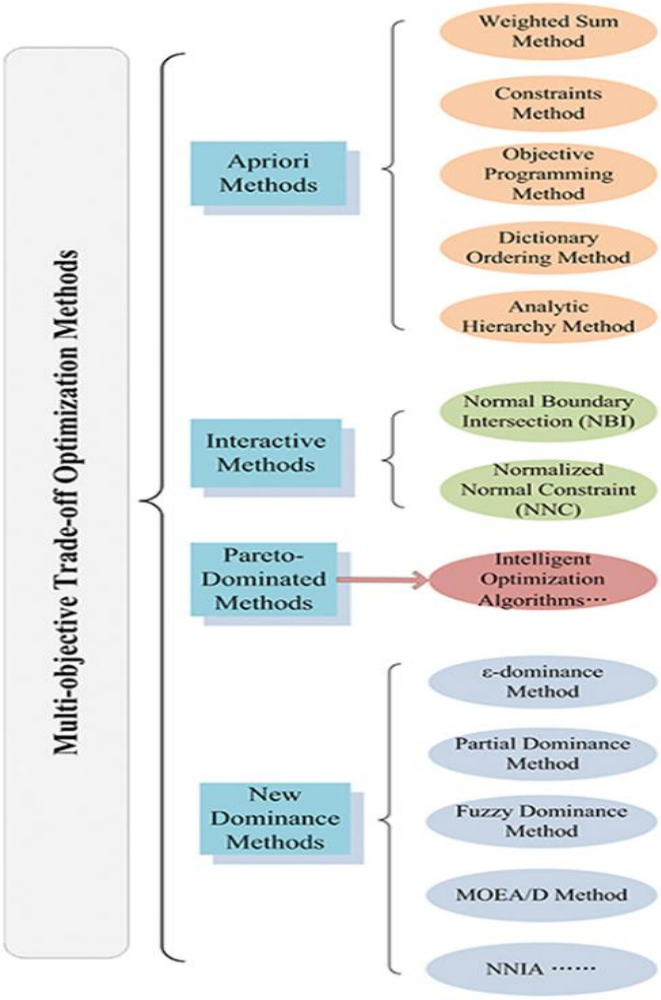

Ilustración 8. Métodos de optimización multiobjetivo (Yunfei Cui 689).

La mayoría de los problemas de optimización de uso real no consisten simplemente en optimizar el valor de una función, sino que involucran varios objetivos a la vez. Este tipo de problemas se denominan Problemas de Optimización Multiobjetivo (MOOP). Generalmente, los métodos de resolución de MOOP hacen uso del concepto **dominancia** para decidir qué solución es mejor en caso de realizar comparaciones.

Dadas dos soluciones s_1 y s_2 para los objetivos f_a y f_b, s1 domina a s2 si:

$$f_a(s_1) > f_a(s_2) \ y \ f_b(s_1) \geq f_b(s_2)$$

O también si:

$$f_b(s_1) > f_b(s_2) \ y \ f_a(s_1) \geq f_a(s_2)$$

Es decir, una solución domina a otra si es mejor o igual en todos los objetivos y al menos mejor en uno de ellos.

Las clasificaciones de los métodos de resolución de problemas multiobjetivo son muy variadas. La más popular es la mencionada por Cohon y Marks en 1975 (Cohon, Jared L. and Marks, David H. 208-220):

- **Métodos a priori**: En esta aproximación dos o más objetivos se combinan generalmente de forma lineal obteniéndose así una única función. Si f_1 y f_2 son dos funciones para optimizar, un método a priori consideraría una única función:

$$F = \alpha * f_1 + (1 - \alpha) * f_2, \qquad con\ 0 \leq \alpha \leq 1.$$

- **Métodos a posteriori**: En este enfoque el objetivo es obtener muchas soluciones con tantos valores asociados como objetivos a optimizar. Dichos valores no siempre son mejores que otros ya que puede suceder que dadas dos soluciones S1 y S2, S1 sea mejor que S2 respecto a F1, pero peor en F2, en tal caso ambas soluciones pueden considerarse igualmente buenas sin tener más información.

Una propuesta diferente es tratar los MOOP con Algoritmos Evolutivos (AE). Los AE son métodos de optimización bioinspirados que toman como modelo la evolución biológica. En ellos, se trata un problema como un entorno en el que los individuos son posibles soluciones al problema que se irán refinando a lo largo del tiempo. Si bien pueden ser utilizados junto a los métodos anteriores, existen AE especializados para tratar problemas multiobjetivo llamados MOEA (Multi-Objetictive Evolutionary Algorithm).

G. Chiandussi a, M. Codegone et al. hacen un análisis de diferentes métodos de resolución de MOOP (Chiandussi 917-921), entre ellos:

- **Método del criterio global (Chiandussi 917)**

El método del criterio global es un método *a priori*. Su finalidad es minimizar una función (criterio global) que mide cuan cerca se encuentra un vector de soluciones del vector de soluciones ideal f_0.

La principal ventaja de los métodos de criterio global es su simplicidad y eficacia ya que no requieren un procedimiento de clasificación de Pareto (clasificación según preferencias). Sin embargo, tienen como desventaja la definición de los objetivos a cumplir, siendo necesario un esfuerzo computacional adicional. Un problema añadido a estas técnicas es que producirán una solución no dominada sólo si los objetivos se eligen en el dominio factible y tales condiciones pueden limitar su aplicabilidad.

- **Método ε-constraint (Chiandussi 919)**

ε-constraint es un método *a posteriori* que consiste en elegir una única función del conjunto de funciones a optimizar y considerar el resto como si se tratasen de restricciones para así conseguir un problema de optimización mono-objetivo con restricciones.

El problema:

$$\min_{x \in X}(f_1(x), \ldots, f_p(x))$$

Pasa a ser:

$$\min_{x \in X} f_j$$

$$sujeto\ a:$$
$$f_k(x) \leq \varepsilon_k, \qquad k = 1, \ldots, p, \qquad k \neq j$$

La principal desventaja de este enfoque es su coste computacional (potencialmente alto), debido al cálculo de los márgenes ε_i. Además, la codificación de las funciones objetivo puede ser extremadamente difícil o incluso imposible para ciertas aplicaciones, especialmente si hay demasiados objetivos. Como ventaja se puede mencionar que es una técnica relativamente sencilla. En la ilustración 9 Z_2 corresponde a una función objetivo tratada como una restricción adicional que delimita la región factible.

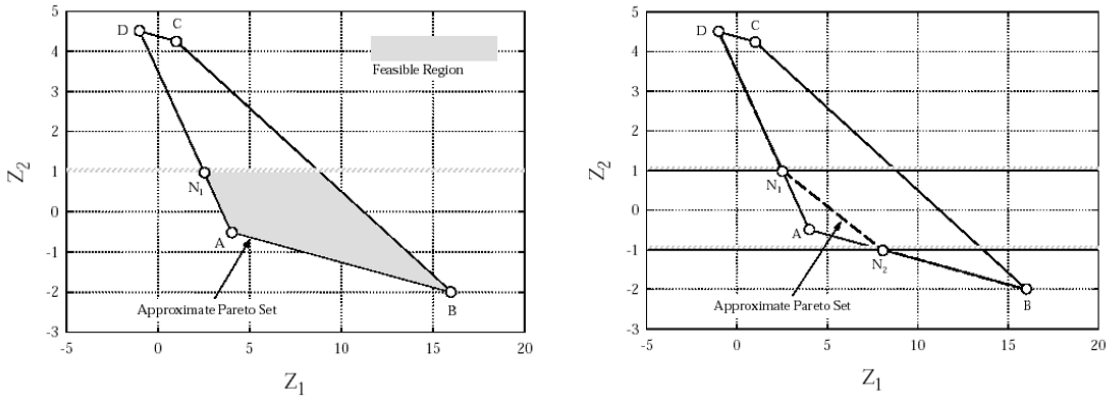

Ilustración 9. Ejemplo del método ε-contraint

- **MOGA**

MOGA es el acrónimo de Multiple-Objective Genetic Algorithm (Algoritmo Genético Multi-Objetivo). Los MOGA intentan encontrar soluciones Pareto-eficientes siguiendo el enfoque de los algoritmos evolutivos.

El concepto Pareto-eficiente o Pareto-óptima proviene del ámbito económico y define una situación en que no es posible beneficiar a una persona sin perjudicar a otra. En el contexto de los MOGA el adjetivo Pareto-óptima se emplea para calificar a soluciones donde variar uno de sus valores para mejorar el valor de una función objetivo implica el empeoramiento de otras. El conjunto de estas soluciones se denomina frente de Pareto. En la imagen 10 los puntos A, B son óptimos que forman parte del frente de Pareto puesto que desplazarlos siguiendo la línea roja tiene como resultado una variación en f1 y f2. C es una solución no óptima.

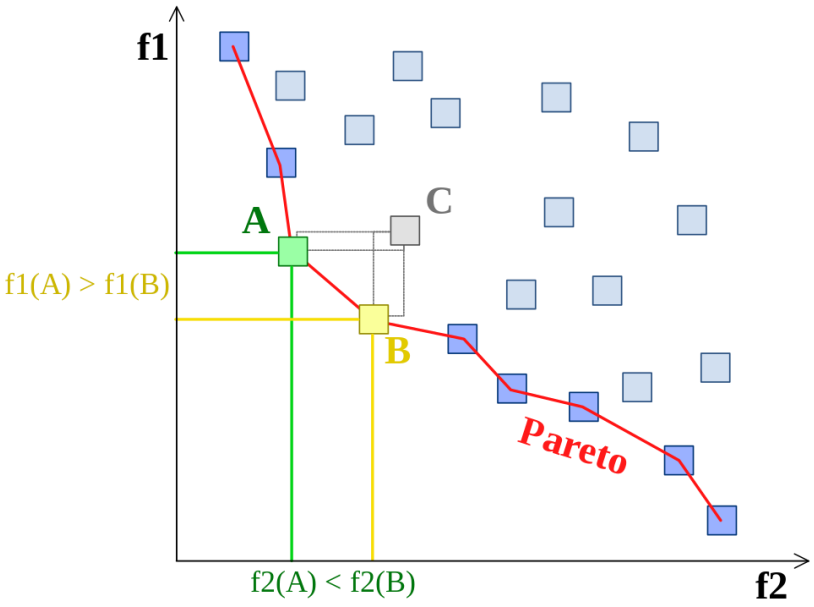

Ilustración 10. Pareto-optimalidad (Wikimedia Commons).

Los Algoritmos Evolutivos (AE) toman como referencia la Teoría de la evolución darwiniana según la cual los individuos más aptos sobreviven y generan descendencia y los menos aptos perecen por lo que tienen menos probabilidades de reproducirse. Su principal ventaja frente a los MOOP es la capacidad de manejar un conjunto de soluciones simultáneamente por lo que es fácil encontrar múltiples soluciones Pareto-eficientes con una sola ejecución del algoritmo. Como principal desventaja se puede mencionar que la codificación de los individuos y el diseño de la evaluación no es trivial.

Las pruebas de G. Chiandussi a, M. Codegone et con cinco referencias mostraron que los MOGA identificaban el frente de Pareto de forma más precisa que los otros métodos. Los métodos criterio global y ε-constraint resolvieron los problemas en un tiempo menor, no obstante, revelaron otras desventajas como la dependencia de la forma del frente de Pareto. Los detalles de los experimentos se pueden encontrar en (Chiandussi 922-941).

4.2. Metaheurísticas

El punto 4.1. pone de manifiesto la dificultad de tratar problemas de optimización de forma exacta debido al elevado coste que requiere hallar la mejor solución. Una forma de sortear esta dificultad es buscar resultados cercanos al óptimo con el fin de encontrar una buena solución dado que en la mayoría de los casos prácticos es suficiente. Además del coste computacional, se encuentra el inconveniente de tratar varios objetivos a la vez requiriendo algoritmos sofisticados que busquen una solución compromiso entre todos ellos. En la sección 4.1. también se ha hecho una revisión de los distintos enfoques seguidos, destacando entre ellos los MOGA.

A pesar desventajas de los MOGA mencionadas en el punto anterior, cuentan con mayores ventajas y cubren los requisitos para tratar el problema por lo que se escogen los algoritmos genéticos para resolverlo. En este apartado se realizan una serie de

definiciones, se explican en mayor profundidad los GA y se describen distintos algoritmos multiobjetivo.

Antes de entrar en la explicación de los GA, es necesario realizar una serie de definiciones para entender la terminología utilizada durante ella.

1. **Gen**: Los genes son unidades de información dentro de los cromosomas. En el contexto de los AG son casillas o nodos dependiendo de la estructura del genoma.

2. **Alelo**: Los alelos son cada uno de los valores que puede tomar un gen, es decir, los que pueden ser asignados a un nodo o casilla.

3. **Genotipo**: Es el conjunto de genes de un organismo. Dentro de los AE suelen ser representados como *arrays*, n-tuplas, listas, etc.

4. **Fenotipo**: En términos biológicos, es la manifestación variable del genotipo de un organismo en un determinado ambiente (Real Academia Española). En los AE es el resultado de evaluar una solución con la función objetivo.

5. **Cromosoma**: Son complejas estructuras formadas por ADN y proteínas. En el marco de los AE se usan indistintamente los términos cromosoma y genotipo.

6. **Operador genético**: Los operadores genéticos son funciones que actúan sobre la población para generar nuevos individuos. Son esenciales para lograr el conjunto inicial progrese mejorando la adaptación de los individuos al entorno. Hay tres principales grupos: operadores de selección, de cruce y de mutación.

4.2.1. Algoritmos genéticos.

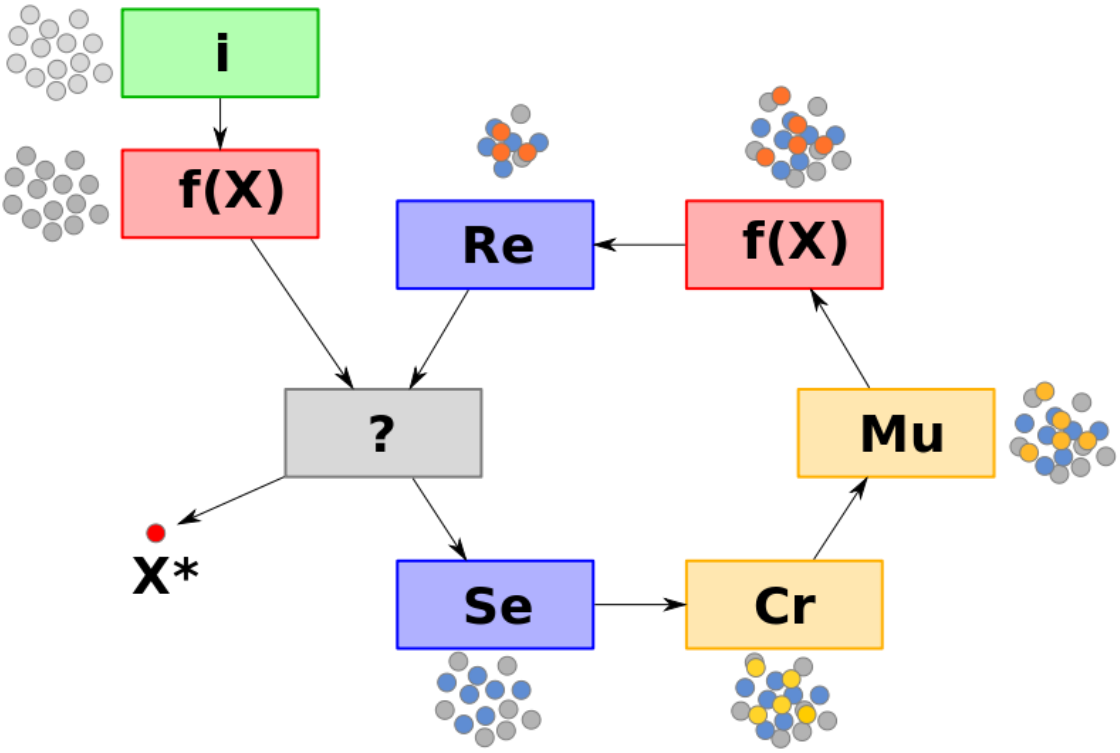

Ilustración 11. Algoritmo evolutivo y sus fases (Wikimedia Commons).

Los algoritmos genéticos (GA) son métodos adaptativos que forman parte de los AE. Los GA pueden ser utilizados en problemas donde se requiera optimización, aprendizaje o búsqueda. Siguen los principios de la evolución propuestos por Darwin en 1859 como la supervivencia de los más aptos y la selección natural.

En la evolución darwiniana, un conjunto de individuos conforman una población inicial que habita un entorno, frente a dicho entorno, un individuo se considera mas o menos apto dependiendo de su capacidad para progresar en él. Al convivir con otros sujetos en el mismo entorno, es inevitable competir por los recursos y es en este contexto donde entra en juego su adaptación. Los seres con alguna ventaja evolutiva frente a la media tienen mayores probabilidades de conseguir los recursos vitales para subsistir y enfrentarse a las dificultades que puedan presentarse en el entorno. Como consecuencia, es mas plausible que estos seres sobrevivan mas tiempo y puedan llegar a reproducirse, transmitiendo aquellas ventajas que le han hecho mejor a su descendencia. Pero, las mejoras que tiene un sujeto sobre otro no son siempre heredadas sinó que aparecen fruto del azar, aquí es donde entran en juego las mutaciones genéticas.

Las mutaciones genéticas son cambios que se generan de forma espontanea en los genes de un individuo. Estas mutaciónes pueden tener tres resultados: Una mejora en el afectado, un empeoramiento o simplemente carecer de efecto. Las mutaciones son algo a tener en cuenta ya que pueden conseguir que un espécimen con una ascendencia en la media llegue a estar entre los mejores.

Como se puede intuir, la población no es una entidad estática, sino que cambia a lo largo del tiempo a medida que sucede la muerte de los individuos, su reproducción y en ocasiones la mutación. Este proceso se puede ver de forma simplificada en la figura 11.

- **Inicialización (i)**: Este es el principio de todo, en este punto se genera la primera población que habitarán el entorno. Los individuos pueden ser muy diversos al igual que su nivel de adaptación al medio.

- **Evaluación (f(x))**: Representa el comportamiento de los especímenes que habitan la población frente al entorno. Estos tienen una serie de características y dependerá de ellas su capacidad para sobrevivir debido a que el medio se encarga de ponerlos a prueba a lo largo de su vida.

- **Finalización (?)**: Indica en qué condiciones termina el proceso de evolución, el fin último de la población. La finalización puede corresponder a sobrevivir un número de generaciones predefinido, alcanzar una determinada adaptación, …

- **Selección (Se)**: El proceso de selección simboliza la competencia de los individuos entre sí en la búsqueda de recursos como puede ser alimento, refugio, compañeros, etc. Los mejores tienen más posibilidades de conseguir lo que necesitan para sobrevivir y avanzar hacia las siguientes etapas de su vida. En cambio, aquellos menos adaptados son más propensos a ser derrotados en la competición y por lo tanto a perecer sin generar descendientes.

- **Cruce (Cr)**: El cruce hace referencia a la reproducción entre dos individuos mediante la cual generan uno o más nuevos que contienen características de cada uno de los progenitores, es decir, los genes hijo son el resultado de la combinación de genes padre. Esta combinación puede producirse de diversas formas dando lugar a múltiples operadores de cruce. El resultado de esta mezcla de genes no siempre da como resultado un ser mejor que sus antecesores.

- **Mutación (Mu)**: La mutación en los genes de los miembros de la población.

- **Reemplazo (Re)**: Los habitantes al cabo de un tiempo envejecen e inevitablemente acaban siendo sustituidos por sus descendientes. Este proceso es necesario para que la población progrese alcanzando mayores niveles de adaptación como resultado de la sustitución de unos individuos iniciales por descendientes que combinan lo mejor de ellos. Aunque el reemplazo suele hacerse de padres a hijos, en el contexto de los AG puede realizarse de distintas formas como por ejemplo de forma aleatoria.

El ciclo evolutivo de un algoritmo genético simple (SGA) se puede resumir en el algoritmo de la ilustración 12:

```
poblacion_inicial = crear_poblacion_inicial(tamaño_de_la_poblacion)
evaluacion(poblacion_inicial)

Mientras no se cumpla la condicion de parada
    hacer:
        seleccionados = selección(poblacion_inicial)
        descendencia = cruce(seleccionados)
        descendencia_mutada = mutación(descendencia)
        evaluacion(descendencia_mutada)
        poblacion_inicial = reemplazar(poblacion_inicial, descendencia_mutada)

fin Mientras
```

Ilustración 12. Algoritmo genético simple.

A partir de este momento queda pendiente concretar para cada problema como se realizará cada uno de los elementos que forman parte del proceso evolutivo.

4.2.2. Algoritmos genéticos multiobjetivo

En este apartado se repasan algunos algoritmos genéticos multiobjetivo que podrían ser empleados para resolver el problema.

Multi-Objective Genetic Algorithm (MOGA).

Es una técnica propuesta por Carlos M. Fonseca y Peter J. Fleming. Esta aproximación asigna un rango a los individuos en función del número de individuos del resto de la población por los cuales son dominados. La aptitud para un miembro de rango n se realiza interpolando las puntuaciones de los individuos con mejor y peor rango.

La principal desventaja de este algoritmo es la dificultad de la coexistencia de soluciones diferentes con valores similares en la función objetivo. Esto se debe a que se motiva que los individuos se encuentren separados respecto a su adaptación. Dado que es habitual desear diferentes individuos que ofrezcan los mismos resultados para poder decidir la más conveniente, no es recomendable si se necesita disponer de múltiples alternativas.

Nondominated Sorting Genetic Algorithm (NSGA).

NSGA es una propuesta de N. Srinivas y Kalyanmoy Deb. Su funcionamiento se basa principalmente en la clasificación de los individuos. Con esta clasificación se persigue la división de las soluciones en bloques. Antes de la etapa de selección, los individuos no dominados se clasifican en un mismo bloque en la cual todos tienen la misma probabilidad de ser escogidos. Después, se crea otro bloque de individuos no dominados ignorando aquellos que se encuentran en el bloque anterior.

La principal desventaja del NSGA es su rendimiento total y, al igual que MOGA, dificulta la presencia de soluciones diferentes con idénticos valores de la función objetivo. Como ventaja se puede mencionar que sus soluciones se ajustan bien al frente de Pareto.

Nondominated Sorting Genetic Algorithm II (NSGA II).

NSGA II surge como solución a los principales problemas presentados por el NSGA, por ello, se centra en la incorporación funciones de coste reducido para conseguir acelerar el algoritmo.

Debido a la intención con la que se desarrolló, cuenta con la virtud de tener un coste temporal relativamente bajo, además, una vez encuentra una región no dominada tiende a proliferar rápidamente. Aun así, tiene una limitada capacidad exploratoria.

Strength Pareto Evolutionary Algorithm (SPEA).

Este algoritmo fue introducido por Eckart Zitzler y Lothar Thiele. Esta aproximación hace uso de un conjunto que conserva los individuos no dominados encontrados hasta la generación actual denominado conjunto externo. Este conjunto externo se rellena con las soluciones no dominadas de cada generación. Para cada miembro de este conjunto, se calcula un valor de fuerza (*strength*) similar al concepto de rango en otros MOGA. En cuanto a la población actual, la adaptación de sus individuos depende de los valores de fuerza de aquellos individuos en el conjunto externo por los que es dominado.

SPEA fue desarrollado para funciones combinatorias continuas por lo que su rendimiento se ve afectado en funciones discontinuas, además el mantener el conjunto externo requiere el uso adicional de memoria.

Pareto Archived Evolution Strategy (PAES)

PAES es un algoritmo desarrollado por Joshua D. Knowles y David W. Corne. Su funcionamiento hace uso de una estrategia de reproducción en la que solo interviene un padre junto a un archivo histórico que contiene algunas de las soluciones no dominadas encontradas.

Como ventaja de este algoritmo se puede mencionar su rendimiento, sin embargo, no tiene un buen comportamiento en frentes desconectados.

Multi-Objective Messy Genetic Algorithm (MOMGA)

MOMGA es una adaptación de los algoritmos genéticos desordenados (mGA) propuesta por David A. Van Veldhuizen y Gary B. Lamont para tratar problemas multiobjetivo. Los mGA siguen una filosofía opuesta a los algoritmos genéticos tradicionales en cuanto a codificación: Frente a los genes de tamaño fijo propone un esquema en el cual el tamaño es dinámico.

Los MOMGA consiguen hallar resultados rápidamente, aunque a costa de un crecimiento exponencial del tamaño de la población y al aumento del tamaño de los cromosomas. Esto supone una desventaja si se pretende consumir la menor cantidad posible de espacio.

Todos los algoritmos presentados tienen sus ventajas e inconvenientes, no obstante, el algoritmo NSGA II resulta especialmente interesante debido a su velocidad de ejecución y el bajo consumo en cuanto a espacio, lo que coincide con el objetivo de conseguir una

solución que ocupe pocos recursos. Asimismo, es uno de los métodos más populares actualmente por lo que resulta sencillo encontrar referencias bibliográficas.

4.3. Diseño de un GA simple

Este punto trata el diseño de los elementos esenciales de un algoritmo genético simple: Los individuos y los operadores genéticos. El punto 4.3.1 detalla el proceso de codificación de una solución en forma de genes, como se decodifican estos y el cálculo de la adaptación del individuo. El apartado 4.3.2. describe el funcionamiento de los operadores genéticos que actúan sobre la población. El punto 4.3.3. especifica la forma en la que dos objetivos pueden ser tratados como uno.

4.3.1. Diseño de los individuos

Los individuos son el elemento central de los algoritmos genéticos, sobre ellos ocurren todos los eventos evolutivos (selección, cruce, …) que llevarán a la población a cumplir el objetivo de su existencia. Son también la conceptualización de las soluciones de un problema pues en sus genes se encuentra el resultado.

4.3.1.1. Codificación de las soluciones

Uno de los primeros pasos a la hora de diseñar un GA es decidir la codificación de la solución en forma de genes de los individuos. En el apartado 4.2 se realizaban una serie de definiciones necesarias para la explicación de los AG. En esta sección se concretan como paso previo al diseño de un individuo.

1. **Genoma o Cromosoma**: El genoma se organiza como una lista en la que existe una o más casillas por cada grado inclinación. El número exacto de casillas depende del número de grados de inclinación y de la precisión:

$$Longitud\ del\ cromosoma \equiv lcrom$$

$$lcrom = \frac{|min.\ inclinación| + |máx.\ inclinación|}{precision} + 1$$

2. **Gen**: Los genes pasan a ser casillas de una estructura de tipo lista.
3. **Alelos**: Los alelos son los valores que se puede almacenar en los genes. Como se ha descrito en la sección 3.1.1., existen valores de presión del pedal entre 0 y 100. Cada valor en ese intervalo corresponde a un alelo.

Con estos tres elementos se tiene un individuo que representa un comportamiento de un conductor frente a las pendientes de la carretera ya que cada gen guarda la información sobre qué nivel de presión del pedal es adecuado ante la inclinación de un tramo.

Un planteamiento diferente de codificación podría ser el considerar un cromosoma en el que cada casilla simboliza un tramo y por lo tanto se establecería una correspondencia directa presión-tramo. Sin embargo, un individuo que funciona bien en una carretera puesto sobre otra diferente tendría un comportamiento inesperado. La dependencia del terreno aumenta ante una situación como la siguiente:

Se tiene un individuo generado con una carretera de 10 tramos de 500 m cada uno. Para situarlo en un terreno de 20 tramos de 250 m se podrían comprimir los 20 tramos sumando las longitudes de dos en dos para tener 10 de 500 m. Una compresión como la mencionada sería viable solo si el ángulo de inclinación fuese siempre el mismo. En caso contrario podría darse la situación en la que dos casillas contiguas, una de ascenso y otra de ascenso se sumasen, degradando así el resultado.

La codificación propuesta establece una dependencia con el grado de inclinación, un dato más acotado puesto que carreteras con elevada inclinación podrían resultar insuperables y aquellas con muy baja serían peligrosas. Esto permite probar un mismo individuo en diferentes autovías sin tener que modificar el individuo, además es de esperar que su comportamiento sea bueno.

4.3.1.2. Decodificación de los genes

La decodificación de los genes corresponde a lo que en términos biológicos se conoce como fenotipo (definición 4 del punto 4.2). El proceso de decodificación se realiza como paso previo a la evaluación del individuo puesto que son sus características las que servirán para cuantificar cuan buena es su adaptación al medio, es decir, cuan buena es la solución al problema.

Debido a que la posición de un gen corresponde a un número entero y los grados de inclinación corresponden al dominio de los números reales, es necesario disponer de una función de mapeo que dada una inclinación calcule el índice del gen en la que se almacena la información sobre el nivel de presión adecuado.

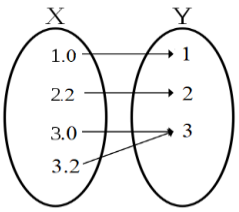

Ilustración 13. Función de mapeo.

$$mapeo(inclinación) = entero\left(\frac{inclinacion - min.inclinación}{precisión}\right)$$

La función de mapeo resta la inclinación argumento entre la mínima de la carretera para asegurar que esta tenga última tenga la posición 0 en el genoma. El resultado de la resta se divide por la precisión para asignar a cada subdivisión de un grado de inclinación una casilla.

Dada una carretera de precisión 0.5 con inclinaciones

$$-2 \quad -1.5 \quad -1 \quad -0.5 \quad 0 \quad 0.5 \quad 1$$

la solución se codificará en

$$\frac{|-2| + |1|}{0.5} + 1 = \frac{2+1}{0.5} + 1 = 7\ genes$$

Cada gen se identifica con un índice entre 0 y 6:

$$0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$$

La información sobre el nivel de presión en un tramo de 0.3 grados se encuentra en el gen

$$entero\left(\frac{0.3-(-2)}{0.5}\right) = entero\left(\frac{2.3}{0.5}\right) = entero(4.6) = 4$$

La decodificación completa de un individuo requiere la información de la carretera y como resultado se obtiene una lista de longitud igual al número de tramos donde cada $elemento_i$ de la lista representa el nivel de presión del pedal en el $tramo_i$.

4.3.1.3. Adaptación del individuo en el entorno

La adaptación de cada individuo se calcula sumando por separado el tiempo y consumo ocupado en cada tramo con la asignación de niveles de presión por tramo obtenida tras la decodificación. Dependiendo del objetivo de la optimización se tendrá en cuenta el tiempo, el consumo o ambos.

A continuación, se describe un ejemplo sencillo para el cual se utilizarán las funciones tiempo y consumo de la sección 3.2.2.2. aunque de forma simplificada.

$$consumo(velocidad\ inicial, tramo, presión\ del\ pedal) = batería\ gastada\ en\ recorrer\ el\ tramo\ argumento$$

$$tiempo(velocidad\ inicial, tramo, presión\ del\ pedal) = tiempo\ en\ recorrer\ el\ tramo\ y\ velocidad\ final$$

Tras la decodificación se tiene que para el tramo 1 se utiliza un nivel de presión 20 y para el tramo 2 un nivel 20. La adaptación se calcula como:

$$Consumo\ tramo\ 1 = consumo(0,1,20) = c1$$

$$Tiempo\ tramo\ 1\ y\ velocidad\ final\ 1 = tiempo(0,1,20) = t1\ y\ v1$$

$$Consumo\ tramo\ 2 = consumo(v1,2,50) = c2$$

$$Tiempo\ tramo\ 2\ y\ velocidad\ final\ 2 = tiempo(v1,2,50) = t2\ y\ v2$$

Este cálculo se realiza de forma similar para todos los tramos. En el ejemplo planteado el tiempo y consumo totales son $t1 + t2$ y $c1 + c2$ respectivamente.

4.3.2. Operadores genéticos

Inicialización

La primera población generada está formada por individuos con cromosomas aleatorios, es decir, se coloca en cada gen de cada individuo un valor de 0 a 100 aleatoriamente siguiendo una distribución uniforme. Esto se hace para abarcar el mayor espacio de búsqueda y en consecuencia aumentar la posibilidad de encontrar valores cercanos al óptimo. El tamaño de la población y la longitud del cromosoma se especifican en el apartado de experimentación.

Evaluación

En esta parte del algoritmo se determina la posibilidad de supervivencia de los individuos en función de su adaptación al problema.

La función de evaluación toma como entradas la población, la suma de las adaptaciones de los miembros de la población y un identificador de la función a optimizar (tiempo, consumo, tiempo y consumo). Si la suma de las adaptaciones es cero significa que ninguno de los miembros cumple las restricciones de tiempo y consumo establecidas. Al ser todos ellos no aptos, se les asigna a todos ellos el valor:

$$1/tamaño\ de\ la\ población$$

Si la suma no es cero, entonces su puntuación se obtiene dividiéndola entre la adaptación de cada individuo:

$$score_i = suma\ de\ las\ adaptaciones/adaptación_i$$

El problema a optimizar consiste en la minimización de objetivos por lo que nos interesan valores pequeños de las funciones objetivo. Dado que la posibilidad de elección de cada individuo se obtiene haciendo un muestreo de los valores de la función objetivo no puede simplemente cambiarse el signo de la función, es por ello por lo que se recurre a una segunda función para revisar la adaptación.

La función de revisión transforma el comportamiento de la función de evaluación haciendo que el lugar de maximizar la adaptación, se maximice la diferencia de la adaptación entre el valor máximo de las adaptaciones, es decir:

$$adaptacion_{nueva} = adaptación_{max} - adaptación_{actual}$$

De este modo, cuanto menor sea el valor de adaptación de un individuo, mayor sea el resultado de la resta y en consecuencia la posibilidad de ser elegido (*score*)

Además de esto, en (Araujo 60) se recomienda aumentar $adaptación_{nueva}$ en un 5% para evitar que la adaptación revisada sea nula si todos los elementos convergen a un mismo valor.

$$adaptación_{nueva} = (adaptación_{max} - adaptación_{actual}) * 0.5$$

La función de revisión se encarga también de comprobar que los individuos de la población no incumplan las restricciones establecidas como el tiempo máximo de trayecto y la capacidad máxima de la batería. A los individuos que superan los límites se les asigna un valor de adaptación 0 y no entran en el cálculo de $adaptación_{max}$. Con la finalidad de acelerar la ejecución del algoritmo se suma la adaptación de los miembros de la población a medida que se recalcula.

El proceso de evaluación queda como sigue:

$$revisar\_adaptación\_minimizar\ (población)$$

$$evaluar\ (población, optimizar)$$

Selección

La selección de los individuos se realiza mediante torneo de pares. Se escogen dos individuos de forma aleatoria y se comparan sus valores *score.* El vencedor pasa a formar parte del grupo de seleccionados para avanzar a la siguiente etapa del algoritmo. El torneo se repite hasta conseguir tantos miembros como en la población original.

Cruce

Para realizar el cruce se siguen tres métodos: Cruce multipunto aleatorio, cruce aritmético y cruce de dos puntos.

- **Cruce multipunto aleatorio**

El cruce multipunto aleatorio parte del cruce multipunto, en el cual los genes hijo se obtienen alternando los genes de los padres, es decir, el primer hijo tendrá el primer gen del primer padre, el segundo del segundo padre, el tercero del primero y así sucesivamente. El segundo hijo obtendrá los genes siguiendo la alternación contraria (segundo, primero, segundo, …).

La diferencia de este cruce radica en la utilización de un valor aleatorio entre 0 y 1 para decidir de que padre se tomará el gen. Si el valor no supera un umbral se toma el gen del primer padre, en caso contrario del segundo.

- **Cruce de dos puntos**

El cruce de dos puntos utilizado generalmente en AG con representación binaria. Este cruce toma dos puntos aleatorios i y j que sirven como índices para establecer que partes del primer y segundo padre pasan a los individuos hijo. Dados i y j, siendo j mayor que i el cruce de dos puntos actúa de la siguiente forma:

Los genes desde 0 hasta i del padre 1 pasan al hijo 1. Ocurre lo mismo con los genes del padre 2 con el hijo 2.

Los genes entre i y j del padre 2 pasan al hijo 1. Sucede de la misma manera con el padre 1 y el hijo 2.

Por último, la información genética entre j y el ultimo gen del primer padre pasan a al primer hijo. El segundo hijo adquiere los genes del padre 2 bajo las mismas condiciones.

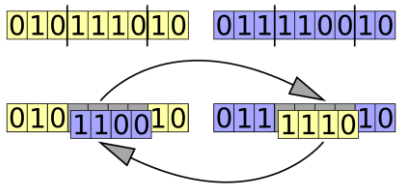

Ilustración 14. Cruce de dos puntos (Wikimedia Commons).

- **Cruce aritmético**

El cruce aritmético (Araujo 110) consiste en realizar la suma ponderada de los alelos de los padres. Para ello, se genera un número aleatorio α entre [0,1] que se utiliza como peso para los alelos del padre 1, en el caso del padre 2 usa como peso $1 - \alpha$. El segundo hijo se genera intercambiado la ponderación.

Dados dos padres p1, p2 con genes [10,10] y [2, 6] respectivamente, con $\alpha = 0.3$ los descendientes generados son:

$$h1 = [0.3 * 10 + (1 - 0.3) * 2, 0.3 * 10 + (1 - 0.3) * 6] = [4.4, 7.2]$$

$$h2 = [(1 - 0.3) * 10 + 0.3 * 2, (1 - 0.3) * 10 + 0.3 * 6] = [7.6, 8.8]$$

Mutación

La función de mutación tiene una gran importancia puesto que de ella depende la capacidad del algoritmo de escapar de óptimos locales. Un óptimo local se da cuando en la búsqueda de una solución se llega a una mejor que otras cercanas, no obstante, puede darse el caso de que existan soluciones con un mejor valor de la función más alejadas. La mutación al convertir un individuo en otro puede producir un "salto" que le permita escapar del óptimo local.

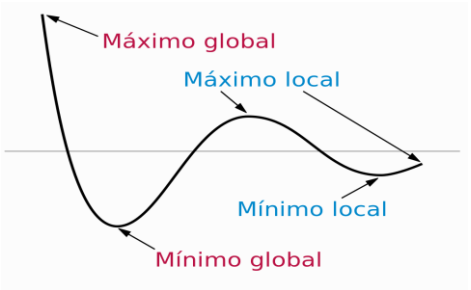

Ilustración 15. Mínimo y máximo global de una función (Wikimedia Commons).

Este procedimiento se realiza mediante mutación uniforme (Araujo 112), según la cual un alelo es sustituido por otro que se encuentre en un intervalo establecido. Para este problema el intervalo empleado es [0,100].

Reemplazo

Para esta parte del algoritmo se escoge el Reemplazo de los individuos peor adaptados. La idea consiste en separar la población en dos subgrupos: El primer grupo abarca aquellos miembros que superan un umbral de adaptación; El segundo grupo contiene los sujetos con adaptación por debajo del umbral y es en este grupo donde se elige quienes serán reemplazados. Una vez definido el segundo grupo, cuando se desee realizar una sustitución se escogerá un individuo de forma aleatoria.

Condición de parada

El algoritmo se detendrá al cabo de un número de iteraciones predefinido.

4.3.3. Tratamiento de múltiples objetivos

Hasta ahora, el algoritmo descrito puede usarse para minimizar solamente el tiempo o el consumo, sin embargo, si se desea optimizar ambos objetivos se ha de buscar la manera de colapsar tiempo y consumo en una sola magnitud.

La combinación de dos o más funciones en una sola puede realizarse de forma sencilla si estas tienen las mismas unidades, para ello, bastaría con realizarse la suma ponderada de todas ellas. Sin embargo, dado que tiempo y consumo tienen unidades distintas esto no puede hacerse directamente. Para solucionarlo, se establece un tiempo y consumo máximo que no ha de ser superado en el trayecto. Con esto, los objetivos pueden tratarse como "utilizar lo mínimo posible del máximo" del siguiente modo: La cantidad utilizada se calcula dividiendo el valor de una función objetivo entre su máximo, siendo el resultado un valor entre 0 y 1:

$$tiempo\ utilizado = tiempo/tiempo\ máximo, \qquad 0 \leq tiempo\ utilizado \leq 1$$

$$batería\ utilizada = batería/batería\ máxima, \qquad 0 \leq batería\ utilizada \leq 1$$

De este modo, un vehículo con límite de tiempo un día cuya batería tiene una capacidad máxima de 100 W que termina un recorrido en una hora gastando la mitad de batería tendría:

$$tiempo\ utilizado = \frac{1h}{24h} = 0.042 = 4.2\%\ del\ tiempo\ máximo$$

$$batería\ utilizada = \frac{50\ W}{100\ W} = 0.5\ = 50\%\ de\ la\ batería\ máxima$$

Si se otorga un peso de 0.5 a cada objetivo, el valor de la función es:

$$0.5 * f_{tiempo} + 0.5 * f_{consumo} = 0.5 *\ 0.042 + 0.5 * 0.5 = 0.271$$

Como se puede observar en el ejemplo, este nuevo enfoque sortea la problemática de las unidades cambiando tiempo y consumo por porcentajes.

4.4. Diseño de un AG multiobjetivo: NSGA II

La presencia de múltiples objetivos en un problema dificulta la categorización de una solución como la mejor de todas, en lugar de ello, tenemos un conjunto de soluciones Pareto-óptimas. El definir una solución como la mejor recae en el usuario, llamado decisor. Es importante ofrecer una amplia cantidad de soluciones Pareto-óptimas para que el decisor pueda elegir la más conveniente a su situación.

El NGSA II es una mejora del algoritmo *Nondominated Sorting Genetic Algorithm* (NSGA). El NSGA es un MOGA y como tal, cuenta con la capacidad de hallar múltiples soluciones Pareto-eficientes en una sola ejecución del algoritmo, sin embargo, es criticado por tres principales motivos:

- **Coste computacional elevado**: El NSGA tiene una complejidad $O(MN^3)$ siendo M el número de objetivos y N el tamaño de la población (Kalyanmoy 1).

- **Falta de elitismo**: El elitismo acelera la convergencia, aunque puede empeorar el comportamiento del algoritmo si las funciones son complejas (Araujo 75).

- **Parámetro de reparto**: Es un mecanismo usado para mantener la diversidad de la población. Representa un cuello de botella en la ejecución del algoritmo (Coello 93).

4.4.1. Procedimientos del algoritmo NSGA II

Con el fin de solventar los inconvenientes del NSGA, NSGA II incorpora la ordenación no dominada rápida y el operador comparación de la distribución.

- **Ordenación no dominada rápida** (*Fast nondominated sort*):

  Para cada elemento p de la población se busca el resto de los individuos q que son dominados por p. Si p domina a q, q pasa a formar parte del conjunto de dominados por p, S_p. Si por el contrario q domina a p se suma uno al contador n_p que representa el número de elementos que dominan a p. En caso de que el contador sea igual a 0 se asigna un rango de uno a p (p_{rank}) y este pasa a formar parte del primer frente, F_1. Una vez definido F_1 se define el resto de los frentes del siguiente modo:

  Se inicializa una variable i a uno.

  Para cada p de F_i se extraen los q de S_p, para cada q se reduce su contador n_q en una unidad y si este llega a 0, se suma uno a su rango q_{rank} y se añade a un conjunto auxiliar Q. Antes de pasar al siguiente elemento p, se suma uno a i y los elementos de Q pasan a F_i. Este ciclo se repite hasta que F_i se encuentre vacío. El coste de realizar esta ordenación es $O(MN^2)$.

```
fast-non-dominated-sort(P)
for each p ∈ P
   S_p = ∅
   n_p = 0
   for each q ∈ P
      if (p ≺ q) then                    If p dominates q
         S_p = S_p ∪ {q}                 Add q to the set of solutions dominated by p
      else if (q ≺ p) then
         n_p = n_p + 1                   Increment the domination counter of p
   if n_p = 0 then                       p belongs to the first front
      p_rank = 1
      F_1 = F_1 ∪ {p}
i = 1                                    Initialize the front counter
while F_i ≠ ∅
   Q = ∅                                 Used to store the members of the next front
   for each p ∈ F_i
      for each q ∈ S_p
         n_q = n_q − 1
         if n_q = 0 then                 q belongs to the next front
            q_rank = i + 1
            Q = Q ∪ {q}
   i = i + 1
   F_i = Q
```

Ilustración 16. Ordenación no dominada (Kalyanmoy 184).

- **Operador comparación de la aglomeración** (Crodwed comparison operator)

El operador de comparación establece un grado de preferencia entre dos soluciones. De las dos soluciones se elige aquella con menor rango, si tienen el mismo rango se elige aquella con mayor distancia de aglomeración. La comparación de todos los elementos tiene un coste $O(NlogN)$.

- **Asignación de la distancia de aglomeración** (Crowded distance assignment)

La distancia de aglomeración es un indicador de proximidad de una solución con otras. Su cálculo requiere la ordenación de N elementos respecto a M objetivos, debido a que los algoritmos de ordenación tienen un coste de N log N el coste de la asignación de la distancia de aglomeración tienen un coste $O(M\ N\ log\ N)$.

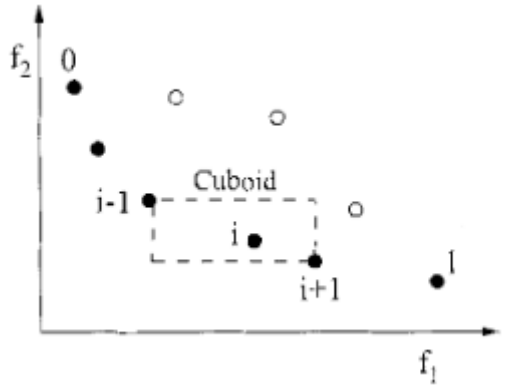

Ilustración 17. Cálculo de la asignación de distancia de aglomeración (Kalyanmoy 185).

Para realizar esta asignación se sigue el siguiente procedimiento:

Se parte de un conjunto de no dominados I.

Se asigna un valor de distancia 0 a cada uno de los miembros i de I.

Por cada objetivo m se ordena I respecto a m y se asigna un valor infinito a los extremos $I_{primero}$ yI_{ultimo}. Para el resto miembros entre el primero y el ultimo su distancia se calcula como la suma entre el valor de distancia previo $I[i]_{distance}$y el cociente de la diferencia entre el valor de m de sus vecinos $I[i+1]_m$, $I[i-1]_m$ y el valor máximo y mínimo de m, f_m^{max} y f_m^{min}.

crowding-distance-assignment$(\mathcal{I})$

$l = \lvert\mathcal{I}\rvert$	number of solutions in $\mathcal{I}$
for each i, set $\mathcal{I}[i]_{\text{distance}} = 0$	initialize distance
for each objective m	
$\mathcal{I} = \text{sort}(\mathcal{I}, m)$	sort using each objective value
$\mathcal{I}[1]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} = \infty$	so that boundary points are always selected
for $i = 2$ to $(l-1)$	for all other points
$\mathcal{I}[i]_{\text{distance}} = \mathcal{I}[i]_{\text{distance}} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$	

Ilustración 18. Asignación de la distancia de aglomeración (Kalyanmoy 185).

Los costes generados son $O(MN^2)$, $O(NlogN)$, $(M\ N\ log\ N)$ siendo la ordenación no dominada rápida la operación que domina el algoritmo con $O(MN^2)$.

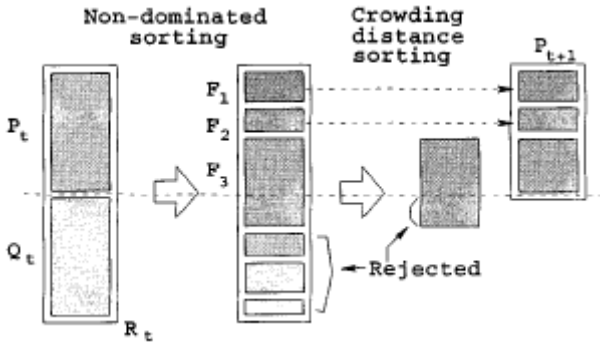

Ilustración 19. Funcionamiento del algoritmo NSGA II (Kalyanmoy 186).

En la ilustración 19 se muestra el procedimiento que sigue NSGA II. Se parte de una población inicial P_t y con ella se genera una población de descendientes Q_t del mismo tamaño para después combinarse en una sola $P_t \cup Q_t$. Este único conjunto se ordena y separa en frentes $F_1, F_2, \dots, F_n$ mediante la ordenación no dominada. Una vez ordenado el conjunto, los frentes se añaden uno a uno en una nueva población P_{t+1}. Si incorporar uno de ellos produce que el tamaño de P_{t+1} supere un umbral (generalmente el tamaño de P_t) dicho frente se ordena con el operador comparación de la aglomeración y se seleccionan la cantidad necesaria para completar P_{t+1}. Este procedimiento se repite hasta que se cumple la condición de parada.

4.4.2. Diseño del individuo

Los individuos en NSGA II seguirán la lógica descrita en el apartado 4.3.1. Las únicas diferencias aparecerán en el momento de la implementación ya que se ha de mantener información sobre los individuos dominados, el número de dominadores y la distancia de aglomeración. Sin embargo, el cómo se guarde esta información no influye en el comportamiento del algoritmo.

4.4.3. Operadores genéticos

El algoritmo NSGA II utiliza componentes del SGA, a excepción de las funciones de revisión de la aptitud y de evaluación. Algunos de los operadores que se incluyen con variaciones son los siguientes:

Selección

Los individuos ordenados mediante el procedimiento de ordenación rápida no dominada son seleccionados mediante torneos tras asignarles una distancia de aglomeración. Se

escogen dos individuos aleatoriamente y aquel con más adaptación es seleccionado para la reproducción.

Reemplazo

El reemplazo sigue el funcionamiento tradicional del NSGA II: La población actual y sus descendientes son ordenados en frentes y posteriormente pasan a una nueva población hasta rellenar un límite establecido (tamaño de la población actual).

5. Implementación

Este capítulo describe la implementación de los diseños del capítulo 3 en el lenguaje de programación Python. El punto 5.1. contiene una descripción del lenguaje de programación utilizado junto a algunas de sus características; también se describen las librerías *NumPy*, *Pandas* y *Matplotlib* que son utilizadas para la implementación y la posterior presentación de resultados experimentales en el capítulo 6. El apartado 5.2. especifica el entorno de desarrollo *PyCharm*. El capítulo termina en el punto 5.3. con la implementación de los algoritmos SGA y NSGA II, describiéndose las clases que representan los modelos del capítulo 3 así como las funciones y procedimientos que intervienen en la ejecución del algoritmo.

5.1. Lenguaje de programación

Python es un lenguaje de programación multiparadigma de código abierto desarrollado en 1991 por Guido Van Rossum. La filosofía de este lenguaje insta a que el código desarrollado sea claro y conciso.

Ilustración 20. Logo de Python.

Dentro de los paradigmas que abarca se encuentra la programación orientada a objetos (OOP) y la programación funcional, de la que toma las funciones *lambda*, *filter* y *map*, entre otras.

Es un lenguaje interpretado, y por lo tanto tiene los beneficios de estos tipos de lenguaje como son el tipado dinámico y la independencia de la plataforma. Python tiene un intérprete que permite programar de forma interactiva facilitando la realización de pruebas y la depuración de fragmentos de código, adicionalmente se puede extender este intérprete con código C si se requiere una velocidad de cómputo crítica.

Al igual que en Java, hace uso de un recolector de basura que ahorra al programador el tener que preocuparse por la gestión de la memoria.

Python es un lenguaje ampliamente utilizado en aplicaciones científicas, siendo algunas de sus principales ventajas la sencillez de su sintaxis y la flexibilidad de sus estructuras de datos como por ejemplo las listas: Mientras que en un lenguaje tradicional como Java o C la combinación de dos listas o seleccionar unos cuantos elementos de ellas requiere el uso de estructuras de repetición (bucles) en Python es suficiente con los operadores **+** y **:**.

En este proyecto se utiliza *Anaconda*. *Anaconda* es una distribución de Python utilizada en ciencia de datos debido a la cantidad de librerías incluidas, estas librerías se pueden gestionar mediante el sistema de administración incluido denominado *Conda*. Entre las

librerías incorporadas se encuentran tres que se usan a lo largo del proyecto: *NumPy*, *Pandas* y *Matplotlib*.

NumPy

Numpy[4] es la contracción de "*Numerical Python*". Entre el contenido de esta librería se encuentran:

- *Arrays* multidimensionales: Pueden ser usados como contenedores eficientes de datos genéricos.
- Funciones eficientes para realizar cálculos numéricos (media, varianza, …).
- Operaciones relacionadas con el álgebra lineal (vectores propios, valores propios, …).
- Funciones para la generación de números aleatorios.

Pandas

Pandas[5] es una librería de código abierto que provee estructuras de datos eficientes y herramientas para análisis de datos. Entre sus características se encuentran:

- Está optimizado con código C.
- Mezcla y unión de conjuntos de datos de alto rendimiento.
- Un objeto *DataFrame* usado para la manipulación de datos, tiene un indexado integrado que se puede personalizar.

Matplotlib

Matplotlib[6] es una librería Python para dibujar gráficos en 2D. Para gráficos simples se hace uso del módulo *pyplot*, que provee una interfaz similar a lenguaje de cálculo técnico MATLAB.

5.2. Entorno de desarrollo

PyCharm[7] es un entorno de desarrollo integrado (IDE) para el lenguaje Python desarrollado por la compañía *JetBrains*. Puede ser utilizado en Windows, Linux y MacOs.

Ilustración 21. Logo de PyCharm.

Entre sus características se encuentran:

- Resaltado de errores sintácticos y autocompletado de código.
- Reestructuración de código.

[4] http://www.numpy.org/
[5] https://pandas.pydata.org/
[6] https://matplotlib.org/index.html
[7] https://www.jetbrains.com/pycharm/

- Posee un terminal integrado que permite utilizar Python en su modo interactivo.
- Herramientas para depuración de código.
- Integración con software de control de versiones (*Git*, *Subversion*, …).
- Extensiones para incorporar otros lenguajes.

5.3. Detalles de implementación

5.3.1. Implementación del algoritmo genético simple

En este apartado se muestran la implementación de algoritmo genético que resuelve el problema planteado en la sección 3 siguiendo las directrices establecidas en el capítulo 4: Diseño y desarrollo. En primer lugar, se encuentran las estructuras básicas usadas durante todo el algoritmo. En segundo lugar, se describen los procedimientos que hacen uso de estas estructuras. Finalmente, se describe la forma en la que dichos elementos actúan con el propósito de obtener una solución al problema.

5.3.1.1. Clases

En un lenguaje de programación las clases son estructuras que representan de forma abstracta objetos del mundo real (aunque no se limita a ellos) encapsulando estado y comportamiento. En este contexto, el estado hace referencia al valor de las variables o atributos y el comportamiento es representado a través de funciones que alteran las variables. Las clases que han sido asignadas con valores concretos se denominan instancias u objetos.

La primera clase presentada es ***TIndividual***. *TIndividual* es el molde que siguen todos los miembros de una población. Para crear un individuo de este tipo es necesario proporcionar solamente la información genética, sin embargo, son necesarios otros atributos para trabajar con él.

x: Fenotipo. Decodificación de los genes de *TIndividual*.

adaptation: Valor de adaptación.

score: Aportación al total a las adaptaciones y en consecuencia la posibilidad de ser escogido en el proceso de selección.

```
1
2  class TIndividual:
3      def __init__(self, genes):
4          self.genes = genes
5          self.x = None
6          self.adaptation = None
7          self.score = None
8
```

Ilustración 22. Definición de la clase individuo en Python.

En segundo lugar, se describe ***Road***. *Road* es una clase que simboliza una carretera, para ello, necesita como parámetros de entrada una lista de secciones (*sections*), la inclinación mínima y máxima del trayecto (*min_slope* y *max_slope* respectivamente) y la precisión. Además, dispone de las variables *n_slope*, *ratio*, *total_len* que guardan el cálculo de la longitud del cromosoma, el número de subdivisiones para cada inclinación y la longitud total del recorrido.

```
1
2  class Road:
3      def __init__(self, sections, min_slope, max_slope, precision):
4          # . . .
5          # Calcular el número de grados de inclinación
6          # Calcular la longitud del cromosoma
7          self.n_slope =  # longitud
8
9          # Número de subdivisiones para cada inclinación
10         self.ratio =  # subdivisiones
11
12         # Sumar la longitud de las secciones para obtener el total
13         self.total_len =  # longitud total
```

Ilustración 23. Definición de una carretera en Python.

Section es cada uno de los tramos en los que se divide una autovía. Una clase *Section* se describe con una inclinación (*slope*), la densidad del aire (*air_density*), la velocidad máxima permitida por señalización (*max_speed*) y su duración en metros (*length*). Estos valores se guardan en atributos del mismo nombre para su uso posterior.

```
1
2  class Section:
3      def __init__(self, slope, air_density, max_speed, length):
4          # . . .
5
```

Ilustración 24. Parámetros de un objeto Section en Python.

Por último, se explica ***Vehicle***. *Vehicle* es la plantilla usada para definir un vehículo eléctrico. Para ello, es necesario introducir múltiples parámetros relacionados con las características técnicas del EV. Los parámetros aparecen en la tabla 4.

Entrada	Tipo	Descripción
drag_area	Valor real	Área de arrastre
mass	Valor real	Masa del vehículo
battery	Valor real	Batería
wheel_radius	Valor real	Radio de la rueda
gear_ratio	Valor real	Caja de cambios
differential_ratio	Valor real	Proporción del diferencial
transmission_eff	Valor real	Eficiencia de la transmisión
max_acceleration	Valor real	Aceleración máxima
speed_torque	Lista de pares (real, real)	Puntos que forman la curva que relaciona la velocidad con el par motor en (m/s, Nm)
speed_power	Lista de pares (real, real)	Puntos que forman la curva que relaciona la velocidad con la energía en (w/s, Nm)

Tabla 4. Parámetros de un vehículo: Tipo y Descripción.

```
class Vehicle:
    def __init__(self,
                 mechanical_efficiency,
                 drag_area,  # m^2
                 rolling_res_coef,
                 weight,  # Kg
                 battery,
                 wheel_radius,  # m
                 gear_ratio,
                 differential_ratio,
                 transmission_eff,
                 max_acceleration,  # m/s**2
                 speed_torque,  # (m/s, N*m)
                 speed_power  # (m/s, watt)
                 ):
```

Ilustración 25. Parámetros de entrada de un objeto Vehicle.

La dinámica del vehículo señalada en la sección 3.2.2.2. se implementa mediante métodos propios de la clase (ilustración 26).

```
class Vehicle:
    def __init__(self,
                 ...

    def fdrag(self, air_density, speed):
    # . . .
    def fslope(self, slope):
    # . . .
    def frolling(self, speed):
    # . . .
    def ftraction(self, speed, pedal_pos):
    # . . .
    def ftotal(self, air_density, slope, speed, pedal_pos):
    # . . .
    def acceleration(self, air_density, slope, init_speed, pedal_pos):
    # . . .
    def pre_time(self, air_density, slope, section, init_speed,
    pedal_pos, section_max_speed):
    # . . .
    def time(self, air_density, slope, section, init_speed,
             final_s, pedal_pos, section_max_speed):
    # . . .
    def final_speed(self, air_density, slope, section, init_speed,
                    pedal_pos, section_max_speed):
    # . . .
    def consumption(self, air_density, slope, section, init_speed,
                    final_s, pedal_pos, section_max_speed):
    # . . .
    def speed_to_torque(self, speed, pedal_pos):
    # . . .
    def time_speed_consumption(self, air_density, slope, section,
                               init_speed, pedal_pos, section_max_speed):
    # . . .

```

Ilustración 26. Funciones de la clase Vehicle.

5.3.1.2. Operadores genéticos

Los operadores genéticos se desarrollan como funciones Python. Estas modifican el estado de las instancias de la clase *TIndividual* haciendo uso de la información proporcionada por la carretera y del EV.

Inicializar población

Initial_population es una función que recibe como argumentos de entrada el tamaño de la población (*pop_size*) y la longitud de los cromosomas (*lcrom*).

La generación de individuos se realiza llamando *a* ***generate_individual*** *pop_size* veces y guardando los objetos devueltos en una lista. Cada llamada a *generate_individual* hace uso del método *random.randint* de la librería *NumPy* para obtener un cromosoma: un *array* de números aleatorios entre 1 y 100 de longitud *lcrom*. Una vez se tiene el cromosoma, se usa como parámetro para crear un objeto *TIndividual*.

Adaptación

El método ***adaptation*** parte de un individuo, un peso w, un vehículo ev y una carretera *road*. Adaptation empieza revisando la información genética y junto a la función *decode* consigue un *array* donde cada casilla corresponde al nivel de presión usado en un tramo. Como último paso se llama a *fun*, una función en la cual cada presión e información de su sección correspondiente pasan al método de ev time_speed_consumption que retorna un tiempo y un consumo. Todos los tiempos y consumos sumados son devueltos por fun y adaptation.

La función **fun** se encarga también de reparar los individuos anómalos producidos por una inadecuada decisión sobre la presión en una inclinación dada. Los errores pueden provocar que un vehículo se quede detenido en medio de un tramo y sin posibilidad de avanzar por lo que el tiempo estimado de llegada puede ser potencialmente infinito. La corrección se realiza aumentando progresivamente la presión del pedal para dicha inclinación en $(100 - presion\ inicial)/10$ unidades

```
1
2   def initial_population(pop_size, lcrom):
3       # generar pop_size individuos
4
5   def generate_individual(lcrom):
6       # generar y devolver individuo
7
8   def adaptation(individual, w, ev, road):
9       # calcular adaptacion
10
11  def decode(genes, road):
12      # genotipo a fenotipo
13
14  def fun(pedal, w, ev, road):
15      # aplicar la funcion al individo
16
```

Ilustración 27. Funciones para inicializar la población y calcular la adaptación.

Evaluación de la población

La función ***evaluate*** recibe como argumentos la población, el EV y un criterio de optimización (*optimization*), el valor por defecto de este último es "*time*" para optimizar el tiempo.

Evaluate recorre todos los elementos de *population*. Como paso previo a la evaluación se llama a *revise_adaptation_minimize* para que *evaluate* otorgue más puntuación a los individuos con menos adaptación y revise si el tiempo límite se ha superado o se ha consumido más batería que la disponible en *ev.battery*.

```
1
2  def revise_adaptation_minimize(population, w, ev, road, optimization="time"):
3      # Revisar adaptaciones
4      # obtener maximo * 1.05
5      # recalcular
6
7  def evaluate(population, sum_adapt=0, optimization="time"):
8      # evaluar
9
```

Ilustración 28. Funciones revision_adaptation_minimize y evaluate

Selección de padres

Selection recibe como argumento la población y su tamaño. Para generar los candidatos a la reproducción se realiza un enfrentamiento entre individuos de la población argumento escogidos de dos en dos de forma aleatoria haciendo uso de la función de Python *random.choice* que recibe como argumentos una lista y un valor que indica el número de elementos seleccionados de ella. Una vez se tienen los dos individuos se comparan sus valores de *score* y el vencedor pasa a una lista de seleccionados. Esta operación se repite *pop_size* veces.

```
1
2  def selection(population, pop_size):
3      # Seleccionar 2 de forma aleatoria
4      # Enfrentar sus score
5      # Devolver todos los vencedores de la comparación
6
```

Ilustración 29. Función selection SGA.

Cruce de los individuos

Crossing toma como entrada dos individuos y el nombre de un operador de cruce. El cruce se realiza accediendo a los genes de los padres y ejecutando el cruce multipunto aleatorio (*multipoint_cross*), el cruce de dos puntos (*two_points_cross*) o el cruce aritmético (*arithmetic_cross*). Como resultado se devuelven dos *arrays* de genes, uno por cada hijo.

- ***Two_points_cross*** hace uso de la función *random.randint* de NumPy para generar dos números aleatorios enteros en el rango [0, lcrom] y con el operador : se escoge el rango de genes:

  $$padre1[0:punto1], \quad padre2[punto1:punto2], \quad padre2[punto2:final]$$

  y *concatenate* de *NumPy* para concatenarlos en uno en uno solo. Para formar un segundo conjunto de genes se intercambian los padres.

- ***Multipoint_cross*** hace uso de un bucle elegir los genes hijo. En cada iteración si un número obtenido mediante *random.random* de la librería NumPy es menor que 0.5 entonces se toma el gen del padre 1 correspondiente al número de iteración, en caso contrario de toma el gen del padre 2.

- ***Arithmetic_cross*** hace uso de *random.random* para producir un número $a \in [0,1]$ que será usado como peso para uno de los padres, el otro tendrá $1 - a$. Los genes y pesos se añaden a la función average de Python para calcular la suma ponderada de los genes. Intercambiando los pesos y volviendo a llamar a average se crea la información genética para el segundo hijo.

```
1
2   def crossing(parent1, parent2, cop="multipoint"):
3       # cruce multipunto, aritmético o de dos puntos
4
5   def multipoint_cross(parent1, parent2):
6       # cruzar padres y devolver dos cromosomas
7       # uno por hijo
8
9   def arithmetic_cross(parent1, parent2):
10      # generar alfa
11      # cruzar padres y devolver dos cromosomas
12      # uno por hijo
13
14  def two_points_cross(parent1, parent2):
15      # se eligen dos puntos de forma aleatoria
16      # se cruzan los genes paternos
17      # se devuelven dos cromosomas
18
```

Ilustración 30. Cruce y tipos de cruce: multipunto, aritmético y de dos puntos.

Mutación

Mutation recibe como argumentos la población, la longitud de un cromosoma, la probabilidad de mutación de un gen (*prob_mut*), ev y *road*.

Para cada miembro de la población se recorre todos y cada uno de sus genes. Si un número generado aleatoriamente para cada gen por *random.random* es menor que *prob_mut*, el valor del gen cambia por un número real en el rango [0, 100] mediante la función *mut_val_double*.

```
1
2   def mutation(population, lcrom, prob_mut):
3       # recorrer genes de los individuos
4       # cambiar genes a mut_val_double si corresponde
5
6   def mut_val_double(max_value):
7       # aleatorio entero entre 0 y 100 (incluidos)
8       # aleatorio decimal entre 0 y 1 (1 excluido)
9       # devolver entero + decimal
10
```

Ilustración 31. Función mutation y mut_val_double.

Reproducción

El proceso de reproducción se lleva a cabo mediante la función ***reproduction.*** La reproducción recibe los candidatos a tener descendencia, aunque hayan sido seleccionados, el generar hijos dependerá de la tasa de cruce definida para el algoritmo. Si un numero generado con *random.random* es menor que la tasa, el individuo es cruzado con otro posteriormente.

Los genes hijo son producidos llamando a la función ***crossing*** con dos padres como argumento y posteriormente dichos genes son usados para crear objetos *TIndividual*. Dado que la población resultante no puede superar a la original, los nuevos descendientes sustituirán aleatoriamente a los individuos con *score* menor que la media ya que serán considerados los peor adaptados.

El proceso evolutivo se encapsula en la función ***search*** que recibe los argumentos indicados en la tabla 5.

Parámetro	Valor	Descripción
Max_gens	Número entero	Número máximo de generaciones
pop_size	Número entero	Tamaño de la población
p_cross	Número en coma flotante en el rango [0,1]	Probabilidad de cruce
p_mut	Número en coma flotante en el rango [0,1]	Probabilidad de mutación
l_crom	Número entero	Longitud del cromosoma
weight	Número en coma flotante en el rango [0,1]	Peso del objetivo tiempo, un valor entre 0 y 1. El peso del consumo se obtiene como 1 – peso del tiempo
cop	Cadena de caracteres: “arithmetic”, “multipoint, “two_points”	Operador de cruce
optimization	Cadena de caracteres: “time”, “consumption”, “time_consumption”	Objetivo a optimizar
ev	Objeto Vehicle	Vehículo eléctrico
road	Objeto Road	Carretera

Tabla 5. Parámetros de la función search.

Search sigue el esquema de un algoritmo genético simple. Sin embargo, consta de dos diferencias: La primera consiste en la adición de la función de revisión de adaptación para realizar la minimización; La segunda diferencia es la "captura" del mejor individuo de todas las iteraciones del algoritmo. Esto se hace tomando el mejor en una generación (devuelto por *evaluate*) y comparando su valor de adaptación para escoger el mejor.

```
1
2  def search(max_gens, pop_size, p_cross, p_mut, l_crom, weight, cop, optimization, ev, road):
3      # Generación de la población inicial
4      # revisar adaptación para minimizar
5      # evaluar la población y obtener el mejor
6
7      # En cada generación
8          # Seleccionar los candidatos a la reproducción
9          # Generación de descendencia
10         # Mutación de los individuos
11         # Recalcular la adaptación
12         # Reevaluar la población y mostrar el mejor
13         # mantener el mejor de todos
14     # Devolver la población resultante y el mejor de todos
15
```

Ilustración 32. Función search de SGA.

5.3.2. Implementación del algoritmo NSGA II

El algoritmo NSGA II implementado se basa en el ejemplo escrito en el lenguaje de programación Ruby escrito por Brownlee (Brownlee 158-163). El algoritmo NSGA II en Ruby se adapta para el lenguaje Python y se añaden los elementos necesarios para el problema concreto que se trata en este proyecto. Asimismo, se reutiliza gran parte del código desarrollado en 5.3.1. por lo que en este apartado se describen los nuevos componentes y algunas diferencias presentes entre GA y NSGA II.

5.3.2.1. Añadidos y diferencias con un GA simple

La primera diferencia la encontramos en la clase que representa al individuo: *TIndividual* pasa a llamarse ***TIndividualNSGAII***, el atributo *x* de *TIndividual* se cambia por ***vector*** y se añaden los atributos *rank*, *dist*, *dom_count* y *dom_set*.

```
1  class TIndividualNSGAII:
2      def __init__(self, genes=None):
3          self.genes = genes
4          self.vector = None
5          self.adaptation = None
6          self.rank = None
7          self.dist = None
8          self.dom_count = None
9          self.dom_set = None
10
```

Ilustración 33. Clase TIndividualNSGAII.

Una segunda diferencia aparece en la función *generate_individual* que pasa a llamarse ***generate_individual_NSGAII*** y genera una lista de objetos *TIndividualNSGAII*.

```
1
2  def generate_individual_NSGAII(l_crom, ev, road):
3      # Llamar repetidamente a generate_individual_NSGAII
4      # Añadir los objetos a una lista
5      # Devolver lista
6
7
```

Ilustración 34. Función para generar individuos

La función *reproduction* pasa a llamarse ***reproduce*** y su ejecución es similar, salvo la llamada a *mutation* de SGA en su interior.

Se añade la función ***calculate_objectives*** que opera de manera similar a *adaptation*: Calcula el valor de adaptación, pero repitiéndolo para todos los individuos argumento.

Se añade ***dominates***, una función que toma dos individuos y compara sus valores de adaptación. Dados dos individuos p_1 y p_2, p_1 domina a p_2 si tiempo y consumo son inferiores en p_1.

Se incluyen las funciones ***calculate_crowding_distance*** y ***crowded_comparison_operator*** que funcionan como se ha descrito en el apartado 4.4.1.

Se incorpora la función ***better*** que compara dos individuos por su rango. El mejor individuo será aquel con un menor rango. En caso de ser iguales se comparan por distancia de aglomeración, siendo mejor tener una mayor distancia.

La función ***select_parents*** recibe una lista de frentes y un tamaño de población. Cada frente se usa para llamar a la función *calculate_crowding_distance.* Después de asignar las distancias, los frentes son añadidos a una lista de individuos inicialmente vacía hasta igualar o superar el tamaño establecido. En caso de superar el tamaño, el frente que lo ha provocado es ordenado con *crowded_comparison_operator* para seleccionar solamente los necesarios para completar la lista

```
1
2  def select_parents(fronts, pop_size):
3      # Calcular la distancia de aglomeracion
4      # Añadir los frentes uno a uno a una lista resultado
5      # Si el añadir un frente puede provocar un desbordamiento, ordenarlo
6      # escoger solo los mejores necesarios para completar el resultado
7      # Devolver la lista
8
```

Tabla 6. Función select_parents

Weighted_sum recibe como entrada un individuo, un peso y un objeto *Vehicle*, Su cometido es sumar el tiempo y consumo de forma ponderada. El peso para el tiempo es el recibido como argumento y para el consumo $1 - peso$. Debido a que no se pueden sumar si se encuentran en diferentes unidades, el tiempo se divide entre 24*3600 segundos (un día) y el consumo entre la capacidad de la batería.

Fast_non_dominated_sort se encarga de organizar la población en frentes. Cada miembro de la población es comparado contra el resto con el operador *dominates.* Los individuos dominados por un individuo p1 se guardan en una lista denominada *dom_set*.

El número de individuos que dominan a p1 se almacena en un contador *dom_count*. Aquellos con *dom_count* igual a cero pasan a formar parte del primer frente.

Los elementos de *dom_set* de cada uno de los miembros del primer frente son revisados y su *dom_count* decrementado puesto que los dominadores pertenecientes a un frente no pueden estar en sucesivos. Cuando el contador de uno de ellos llega a cero pasa a formar parte del nuevo frente. Completado el nuevo frente, los individuos dominados por miembros de este son igualmente revisados y se repiten el resto de las operaciones hasta que no puedan obtenerse más frentes de Pareto.

```
1
2  def fast_non_dominated_sort(pop):
3      # Calcular dominancia de los individuos
4          # Si uno no es dominado por nadie añadir al primer frente
5
6      # El frente actual tiene indice 0
7      # Repetir indefinidamente
8          # Se define un nuevo frente vacío
9          # Revisar dominados
10             # Para cada dominado
11                 # Si su dom_count = 0 forma parte del nuevo frente
12         # actualizar el indice del frente actual
13         # No añadir frentes vacios
14         # Si no se pueden crear mas frentes salir
15     # Devolver frentes
16
```

Ilustración 35. Función fast_non_dominated_sort.

La función ***search*** en NSGA II es similar a la mencionada para el SGA, las principales diferencias radican en la incorporación de la ordenación no dominada antes de la fase de selección y en la desaparición de la revisión de la minimización al no utilizarse el campo *score* en las etapas, también desaparece *evaluate* y en su lugar se emplea *calculate_objectives.*

```
1
2  def search(max_gens, pop_size, p_cross, p_mut, l_crom, weight, cop, ev, road):
3      # Generación de la población inicial
4      # calculate objectives
5      # fast_non_dominated_sort a la población
6      # Generación de descendencia
7      # calculate objectives
8
9      # En cada generación
10         # fast_non_dominated_sort a la población
11         # Seleccionar los candidatos a la reproducción
12         # Generación de descendencia
13         # Calcular objetivos
14         # Mantener el mejor
15
16     # Devolver la población resultante y el mejor de todos
17
```

Ilustración 36. Función search en NSGA II.

6. Experimentación: Evaluación y pruebas

Este capítulo contiene las pruebas realizadas con los algoritmos implementados en la sección 5. En el punto 6.1. se explica las fases que forman el experimento. Los datos necesarios para la experimentación se describen en 6.2. Por último, en el apartado 6.3. se analizan los resultados obtenidos.

6.1. Realización del experimento

Los pasos a seguir para la realización del experimento se pueden dividir en cuatro fases: Creación de un vehículo, creación de carreteras, pruebas previas y comparación de los algoritmos.

Creación de un vehículo

En esta fase se crea una instancia de la clase *Vehicle*.

Creación de carreteras

Para la creación de una carretera se debe especificar inicialmente la longitud de las secciones, la cantidad de estas, la precisión, una lista de velocidades máximas permitidas y una lista de incrementos de inclinación que es usada para evitar cambios brucos de inclinación entre dos tramos.

El elevado número de tramos obliga al uso de un algoritmo para crear una carretera. El algoritmo empleado escrito en pseudocódigo se muestra a continuación:

1. Inicializar una variable inclinación a 0.
2. Repetir hasta conseguir el número deseado de secciones:
   2.1. Escoger una velocidad máxima de la lista de señales aleatoriamente.
   2.2. Escoger un incremento de inclinación de la lista de incrementos.
   2.3. Con los valores de velocidad máxima, incremento, densidad del aire y longitud del tramo crear un objeto *Section*.
   2.4. Sumar a la variable inclinación el incremento

Durante el experimento se realizarán dos tipos de pruebas: Pruebas previas y comparación de los algoritmos. Las pruebas preliminares se realizan con una sola carretera y sirven para comprobar el correcto funcionamiento del algoritmo SGA. La carretera se genera tras inicializar el generador de números aleatorios de NumPy con un valor semilla de 20 mediante *random.seed(20)*. Para comparar el algoritmo SGA con NSGA II se producen 20 vías tras establecer un valor semilla 500 y generar dos *arrays*

de longitud 20 con números aleatorios en el rango [0, 1] invocando a la función *random.random* con valor 20. El primer *array* se utiliza como probabilidades de cruce y el segundo como probabilidades de mutación.

pruebas previas

Esta fase se utiliza para comprobar el correcto funcionamiento del algoritmo SGA optimizando independientemente el tiempo y el consumo. La carretera empleada es la creada en la fase anterior para las pruebas preliminares.

Pruebas de los algoritmos

En esta fase se hace uso de las 20 vías y los dos *arrays* obtenidos para comparar los algoritmos SGA y NSGA II. El funcionamiento de estas pruebas es sencillo: Se escoge un algoritmo y un operador de cruce para optimizar las 20 carreteras hasta probar todas las diferentes combinaciones de operadores y algoritmos.

6.2. Datos del escenario de aplicación

En este punto se proporcionan los datos utilizados para crear los distintos objetos y llamar a las funciones que intervienen en el algoritmo.

Antes de poner en funcionamiento las configuraciones es necesario establecer un valor aleatorio semilla para garantizar la reproducibilidad de los experimentos, con este fin se llama a la función *random.seed* de NumPy con el número 20 para hacer algunas pruebas previas, posteriormente se cambia a 500. El valor de los argumentos se ha escogido sin seguir criterio alguno.

6.2.1. Configuración del vehículo

Los valores concretos del vehículo con los cuales se realizan las pruebas se muestran en la tabla 7.

Parámetro	Valor	Descripción
mechanical_efficiency	0.95	-
drag_area	0.55	-
rolling_res_coef	0.02	Resistencia de las ruedas sobre el asfalto
weight	2200	-
battery	100000	-
wheel_radius	0.33	Ruedas de radio 33 centímetros
gear_ratio	2.65	Proporción de la marcha 2.65:1
differential_ratio	3.2	Proporción del diferencial 3.2:1
transmission_eff	0.8	Eficiencia de la transmisión del 80%
max_acceleration	4.8	-

speed_torque	[(0.0, 426), (18.9992, 426), (24.81072, 325), (30.1752, 274), (37.9984, 175), (44.704, 125), (53.6448, 85)]	Puntos de velocidad-par motor extraídos de la ilustración 3 previa conversión de unidades
speed_power	[(0.0, 0.0), (18.9992, 62.5), (30.1752, 62.5), (37.9984, 48.6111), (53.6448, 34.722)]	Puntos de velocidad – consumo extraídos de la ilustración 3 previa conversión de unidades

Tabla 7. Parámetros del vehículo.

Los puntos de *speed_torque* y *speed_power* se obtienen de la gráfica en la ilustración 3 realizando un cambio de unidades para cumplir las especificaciones de la tabla 3. Para el resto de los parámetros se ha hecho una recopilación de distintas fuentes para conseguir un EV que tenga unos valores razonables, aunque no corresponda a un vehículo real. Ejemplos de estos datos pueden encontrar en las múltiples bases de datos de automóviles existentes[8] [9] [10].

6.2.2. Configuración de la carretera

En la siguiente tabla se muestran algunas variables necesarias durante la creación de la carretera. Las variables pueden tener un nombre cualquiera.

Parámetro	Valor	Descripción
-	-1, 0, 1	Incrementos en la inclinación
-	100, 110, 120, 130, 140	Señales de velocidad
-	100	Número de secciones

Tabla 8. Experimentación: Otros parámetros que afectan a la carretera.

Para definir una carretera, primero es necesario crear los tramos de la componen. Un tramo se necesita como parámetros:

Parámetro	Valor	Descripción
slope	-	Se escoge aleatoriamente del conjunto de incrementos en la inclinación haciendo uso de la función random.choice(lista)
air_density	1.228	Densidad del aire
max_speed	100, 110, 120, 130, 140	Se escoge aleatoriamente del conjunto de señales de velocidad haciendo uso de la función random.choice(lista)
length	1000	Longitud de una sección

[8] http://www.elektriauto.ee/eng
[9] https://www.parkers.co.uk/car-specs/
[10] https://www.ultimatespecs.com/

Tabla 9. Experimentación: Parámetros de un tramo.

Una vez se tienen las secciones se calculan las inclinaciones máxima y mínima. Los datos se utilizan como argumentos del objeto *Road*.

Parámetro	Valor
sections	lista de secciones
min_slope	Mínima inclinación en el conjunto de secciones
max_slope	Máxima inclinación en el conjunto de secciones
precision	1

Tabla 10. Experimentación: Parámetros de una carretera

El procedimiento de creación de una carretera se utiliza para generar 21 carreteras en total, una para la realización de pruebas preliminares y el resto para la experimentación con los diferentes algoritmos. Las 20 carreteras vienen acompañadas del número de secciones con pendientes positivas, negativas e iguales a 0 que las componen.

Carretera	Prob. cruce	Prob. mut.	Inclinación < 0	Inclinación = 0	Inclinación > 0
Prueba	-	-	-	-	-
0	0.694	0.545	53	21	26
1	0.061	0.691	20	13	67
2	0.667	0.831	26	30	44
3	0.559	0.636	80	15	5
4	0.085	0.650	95	5	0
5	0.392	0.954	0	6	94
6	0.497	0.570	13	6	81
7	0.332	0.635	7	4	89
8	0.342	0.036	90	10	0
9	0.366	0.749	96	4	0
10	0.851	0.858	2	9	89
11	0.448	0.485	16	19	65
12	0.316	0.232	12	19	69
13	0.362	0.503	86	10	4
14	0.451	0.136	57	25	18
15	0.915	0.173	77	19	4
16	0.460	0.105	32	4	64
17	0.274	0.244	40	18	42
18	0.917	0.661	0	5	95
19	0.460	0.601	87	8	5

Tabla 11. Carreteras y sus probabilidades de cruce y mutación.

6.2.3. Configuración de los algoritmos SGA y NSGA II

La tabla 12 muestra en valor de los parámetros y el algoritmo que hace uso de él.

Parámetro	Valor	Algoritmo
pop_size	50	SGA, NSGA II
l_crom	Determinado por la carretera	SGA, NSGA II
ev	-	SGA, NSGA II
road	-	SGA, NSGA II
weight	0.5	SGA, NSGA II
optimization	“time_consumption”	SGA
max_gens	300	SGA, NSGA II
p_cross	aleatorio	SGA, NSGA II
p_mut	aleatorio	SGA, NSGA II
cop	“arithmetic”, “multipoint”, “two_points”	SGA, NSGA II

Tabla 12. Parámetros de SGA y NSGA II

La probabilidad de cruce, de mutación y el operador de cruce de los algoritmos SGA y NSGA II se ajustarán de manera experimental para obtener aquella asignación de valores que produzca mejores resultados.

En cuanto al máximo de generaciones y el tamaño de la población, se han elegido números arbitrariamente grandes debido a que hacer ajustes de estos parámetros elevaría el tiempo de cómputo de las pruebas.

6.3. Casos de prueba

En este punto se realiza la comparación de la implementación del algoritmo genético simple con la implementación del NSGA II. Como paso previo se comprueba la correctitud del SGA minimizando tiempo y consumo haciendo uso de niveles de presión prefijados para establecer los valores entre los que debería encontrarse el resultado. Para estas pruebas se utiliza la carretera creada tras asignar un valor semilla de 20.

6.3.1. Pruebas previas: Tiempo y consumo en SGA.

En estas pruebas como valores de *p_cross*, *p_mut* y *cop* se eligen 0.8, 0.4 y “*multipoint*” respectivamente. El objetivo de esta parte es comprobar el correcto funcionamiento del algoritmo, por lo tanto, no se refinan los parámetros escogidos. La carretera utilizada es la primera creada según el punto 6.1.

Tiempo

Para poner a prueba la minimización del tiempo se proporciona una lista en la que cada elemento corresponde a un nivel de presión en un tramo y cada valor es máximo:

Tramo 0: presión 100, …, tramo 99: presión 100 => [100, …, 100]

La lista puede pasarse como argumento de la función *fun* para devolver el tiempo que tarda el vehículo en recorrer la carretera designada.

$$fun([100, \dots, 100], ev, road) = 3011.508, 141235.270\ W$$

El resultado de ejecutar el GA para optimizar solamente el tiempo devuelve un valor cercano:

$$SGA(min.tiempo) = 2990.059\ s, 79494.023\ W$$

Junto al tiempo, **fun** y **SGA** devuelven el consumo generado. Se puede advertir que *fun* retorna un consumo por encima del límite (100 kW) porque no penaliza el máximo permitido. En cambio, SGA al detectar soluciones no válidas y asignarles una adaptación 0, estas acaban desapareciendo dando paso a individuos que saben aprovechar las particularidades del terreno para reducir el gasto eléctrico.

Consumo

Las pruebas llevadas a cabo para el consumo se realizan de forma similar. Sin embargo, en esta ocasión los niveles de presión son 0. Es necesario recordar que a pesar de establecer un nivel 0, *fun* se encarga de corregirlos para llegar al destino sin realizar paradas.

Tramo 0: presión 0, …, tramo 99: presión 0 => [0, …, 0]

$$fun([0, \dots, 0], ev, road) = 7899.061\ s, 33435.013\ W$$

La ejecución del GA para optimizar el consumo devuelve:

$$SGA(min.consumo) = 3542.073\ s, 38048.928\ W$$

A pesar de que la optimización del tiempo y el consumo se realiza de la misma manera, el comportamiento en cuanto al consumo es peor, aunque su valor se mantenga por debajo de la mitad del límite.

6.3.2. Pruebas SGA

Las pruebas realizadas con el algoritmo SGA utilizando los diferentes operadores de cruce para las 20 carreteras ofrecen los resultados de las tablas 13, 14 y 15.

Cruce	Carretera	Tiempo	Consumo	Ambos
Aritmético	0	3140.975	32583.828	0.181
	1	3210.297	50237.710	0.270
	2	4814.890	42223.989	0.239
	3	3159.969	36107.317	0.199
	4	3095.232	44237.032	0.239
	5	3603.136	54179.315	0.292
	6	5077.303	44444.747	0.252
	7	5432.460	50946.878	0.286
	8	3595.770	6368.5539	0.053
	9	3276.718	36243.459	0.200
	10	4787.032	46609.909	0.261
	11	4214.337	42086.321	0.235
	12	5220.828	39612.391	0.228
	13	3230.338	22054.620	0.129
	14	4517.429	29322.727	0.172,
	15	4141.914	12578.365	0.087
	16	3555.040	65116.616	0.346
	17	3474.597	34788.121	0.194
	18	4067.252	58031.825	0.314
	19	3172.413	29909.820	0.168

Tabla 13. SGA con cruce aritmético.

Cruce	Carretera	Tiempo	Consumo	Ambos
Multipunto	0	3138.515	32423.698	0.180
	1	4444.291	42900.712	0.240
	2	5800.234	30365.296	0.185
	3	3612.974	23101.979	0.136
	4	3088.760	38233.958	0.209
	5	4579.274	50732.690	0.280
	6	4709.270	49141.294	0.273
	7	4637.761	43563.687	0.245
	8	3546.011	10518.850	0.073
	9	3295.947	25926.813	0.149
	10	4022.965	50398.253	0.275
	11	3941.547	42644.243	0.236
	12	5486.754	39022.537	0.227
	13	5505.986	13623.718	0.100
	14	4768.400	20992.309	0.133
	15	4419.383	17029.648	0.111
	16	3939.152	56937.829	0.307
	17	4916.576	38134.812	0.219
	18	4785.677	56019.226	0.308
	19	3060.661	38895.130	0.212

Tabla 14. SGA con cruce multipunto.

Cruce	Carretera	Tiempo	Consumo	Ambos
Dos puntos	0	3346.734	29077.416	0.165
	1	4074.209	47808.833	0.263
	2	4745.160	33523.663	0.195
	3	3145.685	38580.486	0.211
	4	3087.512	34982.988	0.193
	5	3975.064	54954.049	0.298
	6	4064.080	49128.190	0.269
	7	3755.144	53871.524	0.291
	8	3492.237	8785.038	0.064
	9	3117.931	44161.022	0.239
	10	4556.408	51664.219	0.285
	11	4076.119	44826.179	0.248
	12	3532.620	53591.940	0.288
	13	3408.055	25772.445	0.149
	14	3800.119	16346.888	0.104
	15	4669.671	19725.522	0.126
	16	3607.512	60286.606	0.322
	17	4679.633	30548.465	0.180
	18	3809.917	63814.659	0.341
	19	3079.531	29002.522	0.163

Tabla 15. SGA con cruce de dos puntos.

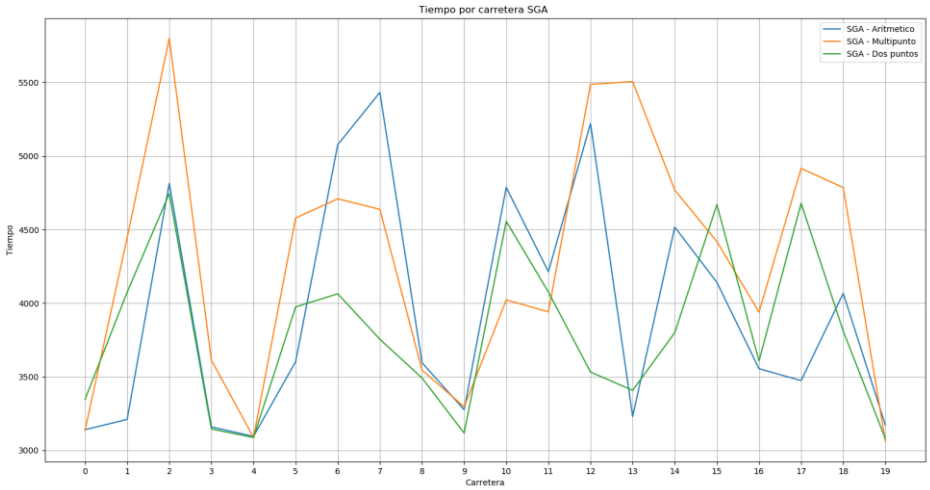

Ilustración 37. SGA: Tiempo en función de la carretera.

La gráfica en la figura 37 muestra los valores de tiempo mínimos obtenidos en cada carretera. Se puede observar que, como es de esperar, en las carreteras predominantemente negativas (0, 3, 4, 8, 9, ...) la duración de trayecto disminuye considerablemente debido al aprovechamiento de las pendientes de bajada para este fin. En casi todas las carreteras, los diferentes operadores de cruce utilizados en el SGA se comportan de forma similar: Disminuyen el tiempo en carreteras predominantemente descendentes y lo aumentan en las ascendentes. La excepción a esta norma aparece en la carretera 13, en la cual el SGA multipunto aumenta el tiempo en una carretera mayormente descendente.

El SGA con el operador de cruce multipunto da menor relevancia al tiempo por lo que los valores que ofrece son más elevados.

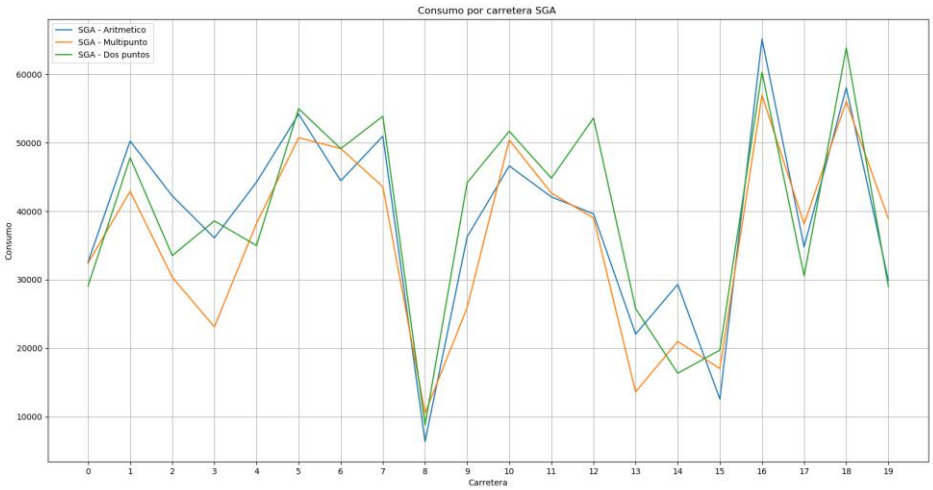

Ilustración 38. SGA: Consumo en función de la carretera.

En la ilustración 38 se puede notar un esquema similar al seguido en la anterior. Lo más destacable es que el SGA multipunto tiene mejores valores de consumo en la mayoría de las carreteras.

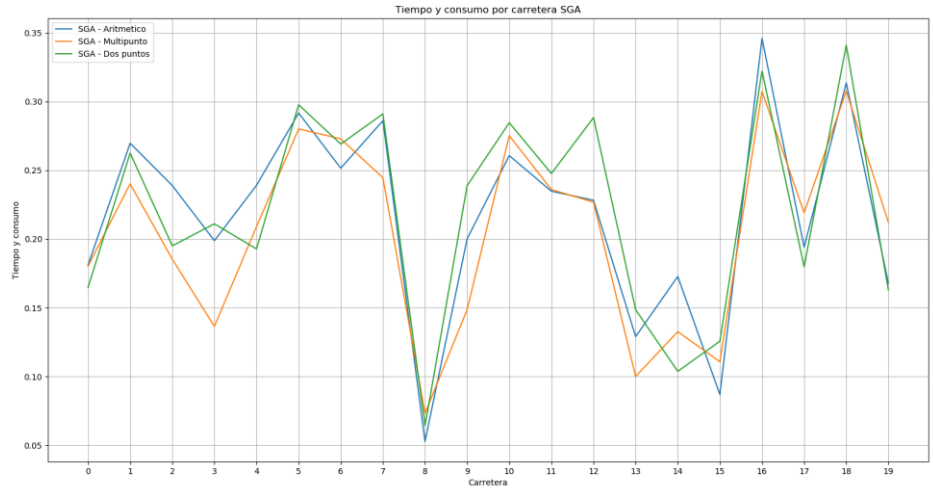

Ilustración 39. SGA: Tiempo y consumo en función de la carretera.

La gráfica de la ilustración 39 combina los valores de tiempo y consumo empleados en las diferentes carreteras (ilustraciones 37 y 38) dividiéndolos por sus máximos y otorgando un peso de 0.5 a cada uno. El operador de cruce que ofrece mejores resultados es el multipunto aleatorio con un valor medio de 0.205 frente a la adaptación de 0.217 y 0.220 de los cruces aritmético y de dos puntos respectivamente.

Aunque las ilustraciones 38 y 39 sean similares, no son exactas: La escala en el eje y es diferente y en la carretera dos de la ilustración 39 el cruce aritmético muestra un ángulo ligeramente convexo.

6.3.3. Pruebas NSGA II

Las pruebas realizadas con el algoritmo NSGA II utilizando los diferentes operadores de cruce para las 20 carreteras ofrecen los resultados de las tablas 16, 17 y 18.

Cruce	Carretera	Tiempo	Consumo	Ambos
Aritmético	0	3842.950	24496.643	0.145
	1	4552.619	37337.539	0.213
	2	5655.923	20014.927	0.133
	3	3152.263	24194.984	0.139
	4	3081.950	25174.021	0.144
	5	5238.081	42305.893	0.241
	6	4938.233	38968.823	0.223
	7	6118.237	40371.615	0.237
	8	3129.711	14743.309	0.092
	9	3238.474	18497.789	0.111
	10	5205.141	41264.885	0.236
	11	5412.485	31169.613	0.187
	12	4811.047	39190.085	0.224
	13	3781.876	12198.201	0.083
	14	4809.865	20102.150	0.128
	15	3336.478	24154.528	0.140
	16	4354.893	50913.075	0.280
	17	4772.888	29182.260	0.174
	18	4617.128	52683.578	0.290
	19	3109.243	25387.080	0.145

Tabla 16. NSGA II con cruce aritmético.

Cruce	Carretera	Tiempo	Consumo	Ambos
Multipunto	0	3719.344	23711.937	0.140
	1	4681.660	34661.564	0.200
	2	4857.767	24174.968	0.149
	3	3224.318	17880.386	0.108
	4	3176.992	21679.231	0.127
	5	5342.165	42665.142	0.244
	6	4211.445	39829.434	0.224
	7	5675.814	37152.932	0.219
	8	3587.127	10091.097	0.071
	9	3231.135	17164.344	0.105
	10	4671.483	43320.383	0.244
	11	4588.578	35605.508	0.205
	12	4547.200	41466.193	0.234
	13	4434.791	9789.0394	0.075
	14	4111.866	15305.072	0.100
	15	4936.765	13222.198	0.095
	16	3882.978	41864.592	0.232

	17	4416.969	31676.606	0.184
	18	5359.497	52339.872	0.293
	19	3085.108	22142.535	0.129

Tabla 17. NSGA II con cruce multipunto.

Cruce	Carretera	Tiempo	Consumo	Ambos
Dos puntos	0	5003.170	22108.862	0.139
	1	4975.934	33979.439	0.199
	2	6019.942	23011.764	0.150
	3	3887.922	24762.819	0.146
	4	3078.852	23065.196	0.133
	5	4562.536	43778.243	0.245
	6	5090.069	38675.166	0.223
	7	5364.278	41691.588	0.240
	8	3496.898	8459.723	0.063
	9	3129.477	17598.074	0.106
	10	4714.316	42915.582	0.242
	11	4399.508	35128.383	0.201
	12	4828.057	39339.649	0.225
	13	3719.395	13836.677	0.091
	14	4963.844	14865.779	0.103
	15	4130.987	12907.448	0.088
	16	4069.043	45299.146	0.250
	17	3419.071	31515.062	0.177
	18	5471.715	49071.254	0.277
	19	3196.038	21958.226	0.128

Tabla 18. NSGA con cruce de dos puntos,

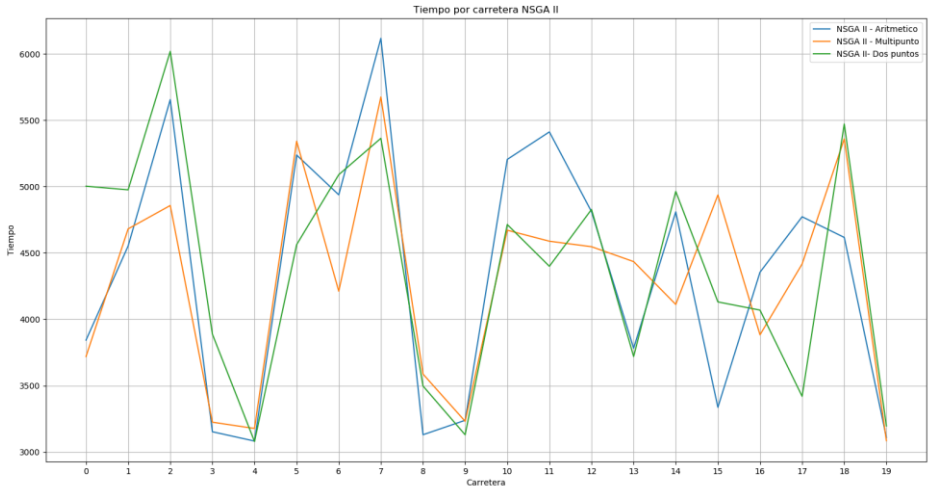

Ilustración 40. NSGA II: Tiempo en función de la carretera.

Los operadores de cruce en NSGA II respecto al tiempo (ilustración 40) siguen un esquema similar a los operadores en SGA (ilustración 37). Sin embargo, el NSGA II con cruce multipunto da menor importancia al tiempo que el mismo cruce en SGA.

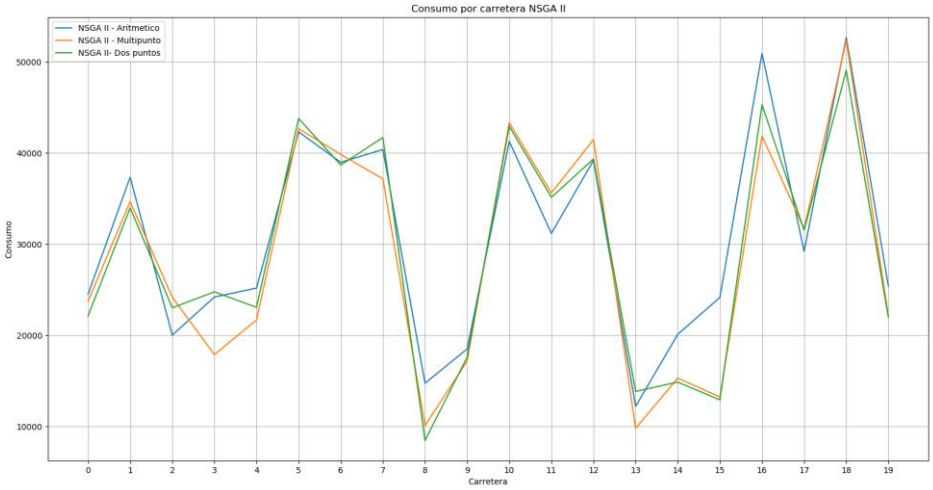

Ilustración 41. NSGA II. Consumo en función de la carretera.

Las diferencias de los operadores respecto al consumo en NSGA II son sutiles por lo que el operador escogido como más favorable depende principalmente de la optimización del tiempo.

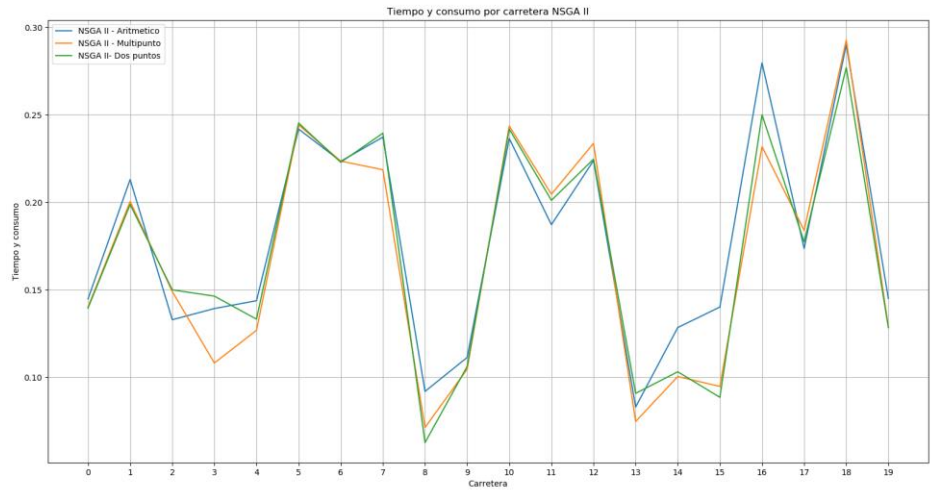

Ilustración 42. NSGA II: Tiempo y consumo en función de la carretera.

La gráfica de la ilustración 42 combina los valores de tiempo y consumo empleados en las diferentes carreteras (ilustraciones 40 y 41) dividiéndolos por sus máximos y otorgando un peso de 0.5 a cada uno. El operador de cruce que ofrece mejores resultados es el multipunto aleatorio con un valor medio de 0.169 frente a la adaptación de 0.178 y 0.171 de los cruces aritmético y de dos puntos respectivamente.

La ilustración 42 es similar a la 41, siendo la única diferencia notoria la escala, esta situación da a entender que el tiempo adquiere poca relevancia en todas las carreteras.

6.3.4. Conclusiones sobre los experimentos

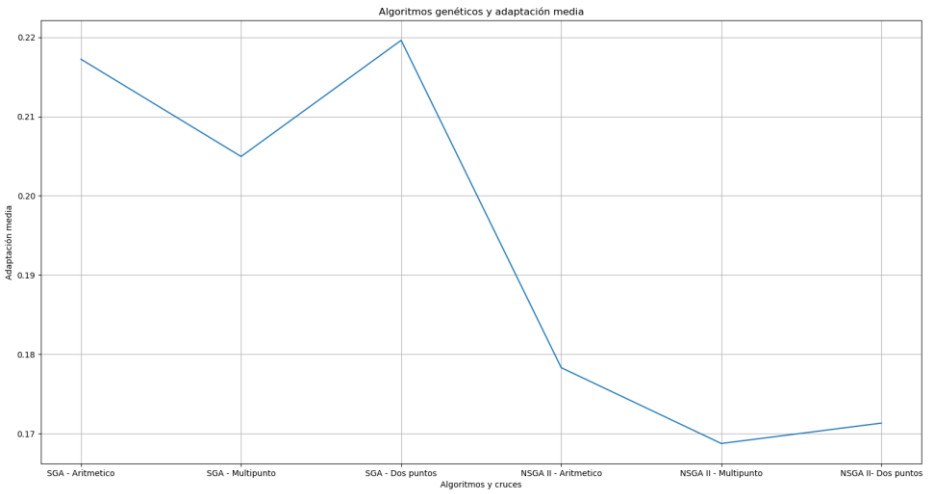

Ilustración 43. SGA y NSGA II: Media de las adaptaciones con los diferentes operadores.

En la figura 43 se observa que los operadores multipunto ya sea para el algoritmo SGA o NSGA II tienen un mejor resultado que el resto. En general, atendiendo solamente al valor medio, el algoritmo NSGA II presenta un mejor comportamiento en la mayoría de las carreteras. Esto se debe a que mientras que NSGA II es un algoritmo especializado para tratar problemas multiobjetivo, SGA mezcla de forma sencilla dos objetivos haciéndolo más sencillo de implementar a cambio de perder precisión.

Los resultados de la imagen 43 son más favorables al operador multipunto puesto que se centra en mejorar el consumo, y según las imágenes 39 y 42, el consumo adquiere más importancia.

Según los datos obtenidos en los experimentos, para conseguir la solución más favorable es recomendable hacer uso del algoritmo NSGA II ya que de media ofrece mejores resultados al optimizar simultáneamente tiempo y consumo, además, NSGA II ofrece un conjunto de soluciones equivalentes debido a su búsqueda de frentes de Pareto por lo que pueden escogerse alternativas en caso de ser necesarias. Sin embargo, la utilización del SGA, a pesar de obtener una buena solución, no garantiza que entre los individuos de la población existan soluciones equivalentes, Por ejemplo:

Cada valor v_i:

$$v_1 > v_2 > \cdots > v_n$$

Podría corresponder a la suma pesada de tiempos (t_i) y consumos (c_i) de forma que:

$$t_1 > t_2 > \cdots > t_n$$

$$c_1 > c_2 > \cdots > c_n$$

Si los valores v_i corresponden a una misma población, esta constaría de individuos que podrían ordenarse según tiempo y consumo sin darse el caso de que $t_i > t_j \, y \, c_i < c_j$. Esto muestra que, si bien es posible obtener una buena solución, no es posible asegurar que se conseguirán alternativas.

7. Conclusiones

Este capítulo se divide en tres secciones: La sección 7.1. Resumen del trabajo realizado. Interés, resultados realiza un breve repaso del problema central de este trabajo junto al interés de su realización. También se justifican las técnicas aplicadas, se repasan los resultados de los experimentos del capítulo 6 y se mencionan algunos detalles de implementación. El punto 7.2. contiene ampliaciones que podrían llevarse a cabo para mejorar el proyecto. Finalmente, en el apartado 7.3. se relaciona la elaboración del proyecto con los estudios cursados.

7.1. Resumen del trabajo realizado. Interés, resultados

Resumen del problema, motivación e interés

El problema que ha motivado el presente trabajo consistía en la minimización del tiempo y consumo en un vehículo eléctrico con la finalidad de obtener beneficios como un mejor aprovechamiento de la batería. Esta es resultado de la adquisición de técnicas de conducción eficiente que pueden dar paso a otros beneficios como por ejemplo el ahorro en un seguro de coche *Pay as you drive.*

Para la resolución del problema se ha hecho uso de un modelo de carretera sencillo en el que el vehículo puede solamente desplazarse hacia adelante siguiendo la inclinación de los tramos de carretera. Un refinamiento de los modelos del vehículo y de la carretera para incluir cambios de dirección junto al uso de un algoritmo que acepte los nuevos modelos, podría dar paso a la elaboración de dispositivos que teniendo en cuenta las características del terreno aconseje al conductor sobre la conducción, incluso sería posible su utilización en vehículos autónomos.

Justificación de las técnicas aplicadas

Las principales dificultades presentes en el problema eran el elevado tiempo de cómputo requerido para buscar la solución junto al inconveniente de tratar con no solo uno, sino dos objetivos que presentaban la dificultad de ser conflictivos debido a que el mejorar el valor en uno de ellos suponía empeorar el valor en el otro.

El método al que se ha recurrido para manejar dichos inconvenientes han sido los algoritmos genéticos: Métodos metaheurísticos que se inspiran en la evolución biológica para encontrar buenas soluciones a un problema. Dentro de los GA se han analizado diferentes algoritmos especializados en problemas multiobjetivo entre los cuales se ha seleccionado el NSGA II.

Los GA resultan especialmente interesantes ya que son capaces de realizar una búsqueda inteligente en grandes espacios de búsqueda suprimiendo el coste que

requeriría probar todas las soluciones a cambio de obtener soluciones que pueden no ser realmente óptimas puesto que quedan regiones de dicho espacio sin explorar. Sin embargo, en problemas reales generalmente basta con buenas soluciones motivando el escoger métodos como los GA.

Respecto al NSGA II, la motivación principal para escogerlo frente a otros métodos han dos: Velocidad y Bajo consumo de memoria. Aunque su coste cuadrático $O(|objetivos| * |población|^2)$ puede ser elevado si se manejan muchos individuos, supone una gran mejora frente a algoritmos como el NSGA (coste cúbico). En cuanto a la ocupación de memoria, el coste dominante se produce al almacenar los individuos de los frentes de Pareto $o(2 * |población|^2 * |genes|)$.

Resumen de las pruebas realizadas y sus resultados

Con la intención de comprobar la capacidad de los algoritmos SGA y NSGA II para hallar buenas soluciones al problema se ha hecho una comparativa en 20 carreteras generadas aleatoriamente para representar distintas situaciones.

Los experimentos realizados han mostrado que el método más apropiado para resolver el problema es el NSGA II con el operador de cruce multipunto ya que ofrece de media mejores valores que el SGA. No obstante, aunque tener valores bajos de tiempo y consumo resulte provechoso, no hay que ignorar que en determinadas situaciones se puede preferir disponer de un algoritmo cuya implementación sea sencilla. Un ejemplo de este tipo de situaciones corresponde al mantenimiento de aplicaciones debido a es más barato analizar y corregir software sencillo.

Notas sobre la implementación

La implementación se ha hecho de forma modular pensando en la reutilización de código por lo que la implementación del NSGA II usa varias funciones del SGA. A pesar de esto, existen más elementos que podrían mezclarse en uno como las clases *TIndividual* y *TIndividualNSGAII*.

El almacenamiento de los individuos se ha realizado con listas simples, lo que aumenta el tiempo de cálculo general del algoritmo cuando se requiere una ordenación ya que se recurre a la función *sorted* de *Python*, este impacto sobre el tiempo de ejecución del algoritmo puede reducirse haciendo uso de estructuras ordenadas. Sin embargo, por motivos de sencillez y claridad del código no se han usado dichas estructuras.

7.2. Posibles ampliaciones

En este punto se mencionan cambios que podrían realizarse para ampliar el proyecto.

El trabajo se ha centrado en optimizar el tiempo y el consumo de un vehículo, sin embargo, sería interesante añadir nuevos objetivos como por ejemplo la comodidad del conductor, ya que este podría no sentirse seguro conduciendo a una determinada

velocidad. En el caso del NSGA II, su implementación está preparada para añadir más objetivos de forma sencilla, siendo la modificación de la función *weighted_sum* el cambio más notable.

La definición de los individuos en el algoritmo genético podría realizarse de tal forma que no solo se tenga en cuenta la inclinación de los tramos, sino también la velocidad a la que se llega a ellos. Su utilidad se puede identificar con la siguiente situación: El vehículo se encuentra en un tramo de ascenso y para superarlo presiona al máximo el acelerador, si inmediatamente después aparece un tramo descendente la velocidad será elevada y podría resultar peligroso para el conductor que se vería obligado a frenar. En principio, esta nueva definición requiere una representación en forma de matriz en la cual las filas podrían diferentes inclinaciones y las columnas velocidades, no obstante, existen técnicas para mapear una matriz como un vector.

Se han probado diferentes probabilidades de mutación, tasas de cruce y operadores de cruce, pero es factible realizar otros cambios como probar diferentes estrategias de selección o incluso el añadir una tasa de mutación variable.

Podría implementarse una interfaz de usuario que facilite la prueba de las distintas configuraciones y que disponga de un conjunto amplio de vehículos y carreteras para experimentar con diferentes modelos.

Por último, se propone incluir un modelo de paralelismo para aumentar la velocidad de ejecución del algoritmo. Ejemplos de paralelismo en GA son los modelos en granja, los modelos de islas y modelos de grano fino o celulares.

7.3. Relación con estudios cursados

El desarrollo del proyecto permite poner en práctica los conocimientos recibidos en diversas asignaturas del grado en ingeniería informática. A continuación, se repasan algunas de las asignaturas que han hecho posible la realización del trabajo.

En el temario *Técnicas, entornos y aplicaciones de inteligencia artificial* se encuentra una unidad didáctica sobre métodos heurísticos y metaheurísticos, dentro de la cual se manifiesta la problemática de la optimización debido a su elevado coste computacional y la necesidad de buscar alternativas, como pueden ser los algoritmos genéticos.

En *Agentes inteligentes* se impartieron conocimientos sobre la Pareto eficiencia y su utilidad para llegar a soluciones compromiso que satisfagan a varias partes.

El concepto de optimización se desarrolla en asignaturas como *Algorítmica* y *Técnicas de optimización.* La primera se centra en el diseño de algoritmos para tratar problemas de optimización junto al refinamiento de su coste temporal y espacial. Los diseños posteriormente se implementan en el lenguaje Python. En *Técnicas de optimización* se explica los ámbitos en los que pueden aparecer este tipo de problemas y se comenta la optimización multiobjetivo.

8. Bibliografía

Amouzgar, Kaveh, y Niclas Strömberg. *Multi-Objective Optimization Using Genetic Algorithms*. Tesis de maestría, Universidad Jönköping, 2012.

Araujo Serna, Lourdes, y Cervigón, Carlos. *Algoritmos Evolutivos : Un Enfoque Práctico*. Editorial Ra-Ma, 2009.

Brain, Marshall. "How Electric Cars Work". Howstuffworks, pp 1-3. https://auto.howstuffworks.com/electric-car.htm (consulta 26/02/2018).

Brownlee, Jason. *Clever Algorithms: Nature-Inspired Programming Recipes*. Primera ed., editorial LuLu, 2011.

Chiandussi, et al. "Comparison of Multi-Objective Optimization Methodologies for Engineering Applications." Computers and Mathematics with Applications, vol. 63, no. 5, editorial Elsevier, 2011, pp. 912–942.

Coello, Carlos A. Coello., et al. *Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition*. Segunda ed., editorial Springer, 2007.

Cohon, Jared L. y Marks, David H. "A Review and Evaluation of Multiobjective Programing Techniques". Water Resources Research, vol. 11, no. 2, 1975, pp. 208–220.

Engineering ToolBox. "Rolling Resistance". *The Engineering ToolBox*, 2008. https://www.engineeringtoolbox.com/rolling-friction-resistance-d_1303.html (consulta 18/01/2018).

Fernández, Jorge Gómez. *A Vehicle Dynamics Model for Driving Simulators*. Tesis de maestría, Universidad de tecnología Chalmers, 2012.

Hidalgo, Jose I., et al. "Un Algoritmo Genético Multi-Objetivo Para La Optimización De Memoria Dinámica En Sistemas Empotrados". Editorial Spanish Ministery of Research (Madrid), 2007.

Hlavacs, Helmut. "A 2D Car Physics Model based on Ackermann Steering". Universidad de Viena, 2006.

Kalyanmoy, Deb, et al. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II". *IEEE transactions on evolutionary computation*, vol. 6, no. 2, 2002.

Konak, et al. "Multi-Objective Optimization Using Genetic Algorithms: A Tutorial". *Reliability Engineering and System Safety*, vol. 91, no. 9, editorial Elsevier, 2006, pp. 992–1007.

Minella, Gerardo, et al. "A review and evaluation of multi-objective algorithms for the flowshop scheduling problem". *Universidad Politécnica de Valencia*, 2017. http://www.upv.es/deioac/Investigacion/Review1_.pdf (consulta 20/03/2018).

López, Juan José. "La pendiente de una carretera - Proyecto Gauss". *Geogebra,* https://www.geogebra.org/m/NSKsdEfY (consulta 13/01/2018).

Luke L. Jensen, Henry Tran, y R. John Hansman. "*Cruise Fuel Reduction Potential from Altitude and Speed Optimization in Global Airline Operations*", 2015.

Petit, and Sciarretta. "Optimal Drive of Electric Vehicles Using an Inversion-Based Trajectory Generation Approach." *IFAC Proceedings Volumes*, vol. 44, no. 1, editorial Elsevier, 2011, pp. 14519–14526.

Receveur, et al. "Multi-Criteria Trajectory Optimization for Autonomous Vehicles". *IFAC PapersOnLine*, vol. 50, no. 1, editorial Elsevier, 2017, pp. 12520–12525.

Python Software Foundation. "The Python Tutorial". *Python*, https://docs.python.org/3/tutorial/index.html (consulta 17/01/2018).

Real Academia Española. "Diccionario de la lengua española". *Real Academia Española,* 23ª ed. 2017 http://dle.rae.es/?id=HlS9odJ (Consulta 23/02/2018).

Real Academia de Ingeniería. "Diccionario Español de Ingeniería". *Real Academia de Ingeniería*, http://diccionario.raing.es/es/lema/densidad-del-aire (consulta 18/01/2018).

Valdes-Dapena, Peter. "Tesla Model S is now the world's quickest car. Yes, Tesla". *CNN Tech*, 2017. http://money.cnn.com/2017/02/07/technology/motor-trend-tesla-acceleration/index.html (consulta 03/03/2018).

Yunfei Cui, Zhiqiang Geng, Qunxiong Zhu, Yongming Han. "Review: Multi-objective optimization methods and application in energy saving". Energy, vol 125, editorial Elsevier, 2017, pp 681-704.