Marketing Computacional: Diseño Automático de Productos

MEMORIA DEL PROYECTO

Stephania Cristina Hinostroza Hualpa
Borja Salazar Rey

Dirigido por Ismael Rodríguez Laguna

Trabajo de Fin de Grado en Ingeniería Informática
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid
Septiembre 2016

Universidad Complutense de Madrid

Departamento de Sistemas Informáticos y Computación

Madrid, España

Marketing Computacional: Diseño Automático de Productos

Memoria para optar al Grado de Ingeniería Informática

Presentada por

Stephania Cristina Hinostroza Hualpa

Borja Salazar Rey

Director:

Ismael Rodríguez Laguna

Autorización de difusión y utilización

Trabajo de Fin de Grado

Marketing Computacional: Diseño Automático de Productos

Curso 2015/2016

Los abajo firmantes, alumnos y tutor del Trabajo Fin de Grado (TFG) en el Grado en Ingeniería Informática de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado (TFG) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Dirigido por Ismael Rodríguez Laguna que pertenece al departamento de Sistemas Informáticos y Computación.

Stephania Cristina Hinostroza Hualpa

Borja Salazar Rey

Ismael Rodríguez Laguna

Agradecimientos

Principalmente nos gustaría agradecer a nuestro director de proyecto Ismael Rodríguez Laguna, por la confianza que ha depositado en nosotros y por los conocimientos que nos ha transmitido a lo largo del desarrollo del proyecto. También queremos mencionar al profesor Pablo Rabanal por ayudarnos a resolver las dudas que hemos tenido en cierto momento. Por último, agradecer a la empresa Feebbo por permitir publicar nuestras encuestas y poder utilizar a sus clientes como encuestados.

Durante estos meses hemos aprendido cómo llevar el proyecto pese a que hemos tenido buenos y malos momentos, sin embargo, nuestro director siempre ha estado ahí para darnos ideas y posibles soluciones que nos ayudado a resolver los problemas nosotros mismos. Por eso agradecemos su implicación con nosotros y el saber decirnos las cosas en cada momento.

En este apartado de agradecimientos queremos mencionar a los pilares más importantes de nuestras vidas y por el cual nosotros estamos aquí, nuestra familia, que siempre nos han apoyado durante estos años de carrera, tanto en los buenos como en los malos momentos que hemos pasado. Así que esto va por y para vosotros, gracias por todo.

Me gustaría mencionar a personas que han estado apoyándome siempre y sobre todo este último año. Una persona fundamental en mi vida y que ha estado día y noche a mi lado cuando lo he necesitado: Omar. A mis amigos de toda la vida que no hace falta mencionarlos porque ellos saben quiénes son. A mis amigos de la universidad: Laura, Samuel, Manu, Eddy, Jorge y Mariano, aquellos con los que he compartido día a día, y que saben en primera persona como han sido estos años de la carrera. Amigos que me han ayudado de alguna forma este último año a base de buenos consejos y de críticas constructivas: Ricardo y Óscar. Gracias a todos vosotros por ser como sois, por formar parte de mi vida y por estar cuando se os necesita.

– Stephania –

Me gustaría hacer mención especial a ciertas personas que durante estos dos años de mi andadura en Madrid han estado cerca de mí, escuchándome, aguantándome, y dándome consejos. Las personas más importantes durante esta etapa, que me ha dado la fuerza cuando lo necesitada y donde siempre he encontrado un apoyo, Susana. Y aunque ya se ha mencionado a la familia, quisiera remarcar a mi hermana Sara, quien siempre es un respaldo y que este año ha conseguido también un gran logro académico. Y a mis amigos de confianza que siempre están ahí llueva o nieve.

– Borja –

Resumen

El objetivo de este proyecto es desarrollar una aplicación multiplataforma que, dadas las preferencias de los clientes por las posibles características que se pueden dar a un producto, y dados los productos que vende la competencia, decida las características del producto a vender para que éste obtenga el mayor número de clientes, bien de manera inmediata, o bien a largo plazo. La solución óptima de este tipo de problemas es intratable, ya que no se pueden resolver en tiempo polinómico, por lo que nosotros utilizamos soluciones heurísticas, concretamente: algoritmos genéticos, algoritmos minimax, algoritmos de aprendizaje automático y algoritmos de interpolación.

Además, realizamos un caso de estudio con datos reales obtenidos a través de una serie de encuestas utilizando una plataforma web, concretamente de la empresa Feebbo, que nos permitió obtener resultados sobre las preferencias de más de 500 encuestados. Las preguntas de las encuestas se centraron en un tipo de producto en particular, en nuestro caso teléfonos móviles.

Palabras clave: *algoritmos genéticos, algoritmos minimax, marketing computacional, Java, Interpolación, Weka, clusters, Feebbo, PSO, SA.*

Abstract

The goal of this project is to develop a multiplatform application that, given the preferences of customers for the possible features that can be given to a product, and given the products sold by the competition, decides the characteristics of the product to sell to get the largest number of customers either immediately or over time. Finding the optimal solution of such problems is intractable and thus it cannot be solved in polynomial time, so we use heuristics solutions, namely: genetic algorithms, minimax algorithms, machine learning algorithms, and interpolation algorithms.

Also we carry out a case study with real data that were obtained through a series of surveys using a web platform, specifically Feebbo, allowing us to obtain results on the preferences of more than 500 respondents. The survey questions focused on a particular type of product, in our case is mobile phones.

Keywords: *genetic algorithms, minimax algorithms, computational marketing, Java, interpolation, Weka, clusters, Feebbo, PSO, SA.*

Índice

Índice de Figuras

1. Introducción

En este primer apartado de introducción describiremos brevemente cómo estará estructurada la memoria. Además, explicaremos los objetivos que tenemos planteados para realizar el proyecto de Marketing Computacional: Diseño Automático de Productos.

1.1 Organización de la Memoria

La memoria está estructurada de la siguiente forma:

- En el primer capítulo, se exponen los objetivos del proyecto.
- En el segundo capítulo describiremos los problemas que abordará la aplicación.
- En el tercer capítulo abordaremos el estado del arte. Mencionaremos las actividades desarrolladas por diversas empresas existentes en la actualidad, relacionadas con estudios de mercado.
- En el cuarto capítulo presentaremos cómo hemos llevado a cabo el diseño de nuestra aplicación.
- En el quinto capítulo describiremos las funcionalidades que tiene nuestra aplicación tanto para escritorio como para Android.
- En el sexto capítulo mencionaremos qué herramientas hemos elegido para realizar la implementación y los resultados que ofrecen.
- En el séptimo capítulo se comentará la implementación de la aplicación multiplataforma.
- En el octavo capítulo describiremos las encuestas que hemos realizado en la plataforma Feebbo, así como el tipo de preguntas realizadas y los resultados obtenidos.
- En el noveno capítulo presentaremos los resultados obtenidos en nuestro caso de estudio de la aplicación.
- En el décimo capítulo daremos las conclusiones finales al desarrollar este proyecto. También discutiremos las posibles extensiones y mejoras que se pueden dar a este proyecto.
- En el undécimo capítulo comentaremos el apartado de trabajo futuro, describiendo las diversas ampliaciones que se pueden hacer al proyecto.
- El duodécimo capítulo presentaremos la organización del desarrollo del proyecto y la contribución que cada uno hemos hecho cada miembro del grupo.
- En el decimotercer capítulo será la bibliografía utilizada.

- En el decimocuarto capítulo se hará un resumen de los objetivos, conclusiones y trabajo futuro en inglés.

1.2 Objetivos

Los objetivos principales del proyecto son:

1. Desarrollar una aplicación que, a partir de las preferencias de los clientes sobre las características de un producto específico, y teniendo en cuenta los productos ofrecidos por la competencia, obtenga el producto ideal de forma inmediata o a largo plazo, es decir:

    a) Diseñar un producto totalmente desde cero, para maximizar el número de clientes o para llegar a un número determinado de clientes; o bien
    b) Diseñar una estrategia que indique cómo iremos modificando nuestro producto a medida que la competencia vaya modificando los suyos, de forma que obtengamos un número de clientes en un plazo determinado independientemente de los cambios que hayan realizado los competidores.
    Se considera también una versión alternativa del problema anterior en la que se restringe la fecha límite: el número de turnos no puede ser mayor que el número de atributos de productos.

Nuestra aplicación resolverá varias variantes de dichos problemas, y lo hará permitiendo aplicar diversos algoritmos diferentes. Además, ofrecerá tanto una versión de escritorio para Windows como una versión de móvil para Android.

2. Vamos a demostrar la utilidad de la aplicación llevando a cabo un caso de estudio realista completo.
3. Finalmente compararemos los diversos algoritmos en la resolución de diversas variantes de los problemas planteados.

Para lograr estos propósitos, se requiere llevar a cabo las siguientes actividades:

- Profundizar en nuestros conocimientos sobre algoritmos genéticos, algoritmos de optimización por enjambre de partículas, algoritmos de enfriamiento simulado, algoritmos minimax y algoritmos de interpolación para aplicarlos en el desarrollo de nuestra aplicación y de nuestro caso de estudio.
- Diseñar y llevar a cabo un conjunto de preguntas para las encuestas que permitan, a través de la página web de *Feebbo* [1], obtener resultados reales de las preferencias de los encuestados sobre un producto en concreto, en nuestro caso teléfonos móviles.

- Aprender distintos algoritmos de clusterización, para clasificar a los encuestados en clusters organizados por similitud.
- Diseñar nuestra aplicación, indagar qué herramienta podemos utilizar para que sea multiplataforma, y por último las funcionalidades que queremos abarcar.
- Desarrollar e implementar la aplicación que nos sirva para diseñar el mejor producto.
- Mejorar la habilidad de programación en el lenguaje Java y aprender a usar funciones contenidas en sus librerías.
- Evaluar los resultados de la aplicación y analizarlos en función de los productos obtenidos.
- Elaborar una documentación que recoja todo lo que hemos realizado y que permita en un futuro ampliar el proyecto.

2. Definición del Problema

(a) El problema *Product Design Optimization* (PDO) consiste en lo siguiente. Supongamos que nos dan un conjunto de atributos de productos (por ejemplo: color, forma, etc.), los subconjuntos de valores de atributos disponibles para cada competidor, los productos que vende cada productor y las valoraciones de los valores de cada atributo puntuadas por los clientes de cada perfil (por ejemplo: cierto perfil lo valora 3 sobre 5 que sea rojo). El objetivo es seleccionar los valores de los atributos de nuestro producto (por ejemplo: verde, cuadrado, etc.) de tal manera que se maximice el número de clientes o se consiga un número determinado de clientes.

(b) El problema S*ucceed in the Product Design Game* (SPDG) consiste en lo siguiente. Supongamos un juego en el que cada productor modifica por turnos algunas características de su producto, y entonces todos los clientes vuelven a escoger sus productos preferidos. Nos dan como parámetros de entrada para el juego: el número de atributos que cada productor puede modificar de su producto en cada turno, el número de turnos del juego y el número de turnos previos entre los que se calculará la media de clientes obtenidos por cada productor. La configuración de cada juego define en el turno actual el producto vendido por cada productor, y el número de clientes conseguidos por cada productor en cada uno de los últimos turnos del juego. El objetivo es encontrar una estrategia que garantice que cierto productor alcanzará cierto número determinado de clientes promedio durante un cierto período de tiempo independientemente de las estrategias de sus competidores.

(c) El problema Linearly-Bounded SPDG (lSPDG) es idéntico al SPDG, aunque el número de pasos del juego está limitado por el número de atributos.

En el artículo [2], el director del presente trabajo y sus dos coautores demuestran que estos problemas son NP-completo, EXPTIME-completo, y PSPACE-completo, respectivamente, lo que demuestra que encontrar su solución óptima es intratable (es decir, que no pueden ser resueltos en tiempo polinómico) bajo suposiciones estándar.

Aunque la resolución de estos problemas de manera óptima para casos muy pequeños puede ser factible, es claramente inviable para la mayoría de los casos con tamaños mayores.

Esta dificultad obliga al uso de soluciones heurísticas para resolver estos problemas. Por desgracia, ni siquiera el problema más simple (a), puede ser bien aproximado en tiempo polinómico en general, ya que además de ser NP-completo, también es Poli-APX-completo, cómo también se demuestra en [2]. Esto implica que no puede existir un algoritmo polinómico que dé soluciones aproximadas cuya ratio con las soluciones óptimas alcancen siempre cierta constante.

La definición y la dificultad de dichos problemas se ven afectadas por el modelo que representa la preferencia de los perfiles de los clientes sobre los productos. Consideramos que, para cada perfil de cliente, los clientes de dicho perfil asocian una valoración (un entero positivo) a cada valor de atributo. Los clientes eligen el producto cuya suma de valoraciones de los valores de todos sus atributos sea la más alta (para algunos clientes reales, algunas características podrían ser negativas en lugar de positivas, aunque un modelo que permite valoraciones negativas se puede convertir trivialmente en un modelo equivalente que use solo valoraciones positivas).

Alternativamente, supongamos que las preferencias del cliente se definen por medio de las funciones más generales que, dados los valores de los atributos de todos los productos disponibles, devuelve el producto preferido. Supongamos que las únicas condiciones impuestas sobre estas funciones sean estas:

(i) Las preferencias son transitivas (es decir, si se prefiere A sobre B y B en C, entonces preferimos A sobre C).
(ii) Las funciones que devuelven el producto seleccionado se pueden calcular en un tiempo polinómico.

En nuestro caso, las nuevas versiones de los problemas serían generalizaciones triviales de nuestros problemas originales, por lo que mantendrían su NP y Poli-APX-dureza, EXPTIME- dureza, y PSPACE- dureza, respectivamente.

3. Estado del Arte

A continuación, comentamos diferentes empresas que realizan una investigación sobre estudios de mercado, a través de las preferencias de sus clientes sobre un amplio número de productos. Casi siempre este tipo de empresas realiza estos estudios para venderlas a compañías más grandes que están interesadas en los resultados que han obtenido, y ofrecen una bonificación a los sujetos encuestados para hacer su estudio.

- Consumolab [3]: es un centro dedicado a la investigación y estudios de las preferencias de consumo. Emplea *técnicas de análisis y marketing sensorial* para comprender las preferencias de los consumidores sobre un producto y qué se puede hacer para dirigir su éxito en el mercado.
  Además, ofrece diferentes tipos de estudio:
  1. **Marketing:** estudio de competidores, caracterización del producto ideal, selección del mejor prototipo entre varios, investigación de la segmentación de las preferencias (clusters), etc.

  2. **I+D:** desarrollo e innovación de productos, etc.

  3. **Calidad:** tiempo de vida útil del producto, etc.

- PanelSilliker [4]: Es una compañía internacional que ofrece servicios para la mejora de la calidad y seguridad alimentaria. Llevan a cabo estudios de mercado para conocer la opinión de los consumidores acerca de una multitud de productos. Estos estudios se realizan a través de *servicios de análisis y análisis sensorial*.
  El propósito de estos estudios es que las valoraciones de los consumidores ayuden a los fabricantes a mejorar sus productos.
- MySurvey [5]: es un panel de consumidores formado por personas que se ofrecen voluntariamente para participar en estudios de investigación de mercado.
- SurveyMonkey [6]: Empresa que permite crear a los usuarios encuestas en línea y analizar los datos. Permiten el análisis de texto, integración SPSS y generar informes profesionales.
- Feebbo [1]: Es una empresa internacional que permite crear estudios de mercados y seleccionar un número personas para que participen en las encuestas. Se pueden ver los resultados en tiempo real y en diferentes formatos: Excel, SPSS, etc. Posteriormente, esta empresa se mencionará más en detalle, debido a nuestra colaboración con ellos.

La mayoría de estas empresas utiliza la herramienta *Fizz* para realizar sus encuestas, ya que les permite obtener de forma eficiente el análisis sensorial de los alimentos. Este análisis tiene como finalidad evaluar atributos de productos a través de los diferentes sentidos.

La ventaja de utilizar esta herramienta es que aparte acelerar y automatizar los análisis, reduce de modo considerable el tiempo de análisis de la información suministrada por los consumidores y aporta nuevos mecanismos para tomar mejores decisiones.

Estos análisis de datos se basan en realizar protocolos de pruebas, capturar respuestas de los consumidores de manera automática, realizar estadísticas y gráficos.

Las empresas mencionadas extraen las preferencias de los usuarios sobre diversos aspectos, pero no utilizan dicha información para tratar de componer automáticamente el producto que obtendría más clientes. Es en dicho segundo paso en el que se centrará nuestra aplicación.

4. Diseño de la Aplicación

En este apartado explicaremos la arquitectura de nuestra aplicación, así como las decisiones de diseño que hemos tomado.

Desarrollar nuestra aplicación implica hacer frente a dos problemas conceptualmente diferentes (ya que SPDG y lSPDG sólo se diferencian en el número de turnos). En PDO, se crea un producto desde cero que maximice el número de clientes. Nuestra aplicación resolverá este problema mediante el uso de un algoritmo genético, un tipo de algoritmo que se puede aplicar a una gran variedad de problemas de optimización NP-duros.

Además, se implementan otros algoritmos de optimización que son alternativos al algoritmo genético: optimización por enjambre de partículas y enfriamiento simulado. Ambos resolverán también el problema PDO. La finalidad de incluir estos algoritmos es observar los resultados que ofrece cada uno y compararlos.

En SPDG, se pretende maximizar el número de clientes medios a largo plazo dentro de un sistema de turnos (con su variante lSPDG, que restringe el número máximo de turnos). Este problema se resolverá mediante un algoritmo minimax con poda alfa-beta, que es un método comúnmente utilizado para resolver problemas de juego EXPTIME-duros y PSPACE-duros.

Tomando como referencia inicial las versiones básicas de ambos algoritmos que realizó el profesor Pablo Rabanal para llevar a cabo los experimentos expuestos en [2], estructuramos el código de nuestra implementación de la siguiente manera:

- El proyecto *Product Design* está organizado en distintos paquetes:
  a) Paquete General: almacena las clases comunes para todos los algoritmos y una interfaz que implementará cada algoritmo.
  b) Paquete Genetic: representa la implementación del algoritmo genético. Contiene una clase abstracta del algoritmo genético, y otra clase que la implementa.
  c) Paquete Minimax: representa la implementación del algoritmo minimax. Contiene una clase abstracta del algoritmo minimax, y otra clase que la implementa.
  d) Paquete PSO: representa la implementación del algoritmo de optimización por enjambre de partículas. Contiene una clase abstracta del algoritmo PSO, y otra clase que la implementa.

e) Paquete SA: representa la implementación del algoritmo de enfriamiento simulado. Contiene una clase abstracta del algoritmo SA, y otra clase que la implementa.
f) Paquete Input: para las clases que permiten leer los datos de entrada.
g) Paquete Output: contiene clases que almacenan los resultados.
h) Paquete GUI: que almacena la clase que implementa la interfaz para escritorio.
i) Paquete Main: representa la clase principal del proyecto de Java.

- El paquete General engloba las siguientes clases:
  a) *Attribute*: representa las características que se pueden asignar a un producto.
  b) *CustomerProfile*: representa la estructura de un perfil de cliente. Contiene una lista de atributos y cada perfil tiene una lista de subperfiles.
  c) *Producer*: representa a cada uno de los competidores. Un productor tiene una lista de atributos disponibles y un producto.
  d) *Product*: representa a cada individuo de la población. Cada producto tiene una lista de atributos y su valor asignado.
  e) *Interpolation*: contiene los métodos que permiten estimar el precio de un producto.
  f) *LinkedAttribute*: representa la implementación de una variante de nuestros problemas en la que ciertas combinaciones de valores de atributos suponen una modificación positiva o negativa de la valoración total del producto si van juntos. Esto permite representar situaciones donde las preferencias de los clientes puedan ser super-aditivas o infra-aditivas, lo que no estaba contemplado en los problemas originales.
  g) *Problem*: esta clase contiene la implementación de los métodos que son comunes para cada uno de los algoritmos. La finalidad de crear esta clase es abstraer los métodos para facilitar la agregación de un algoritmo nuevo si así se requiere en un futuro, además de favorecer la reutilización de código y optimizar la implementación de la aplicación.

  - Los paquetes que hacen referencia a cada algoritmo contienen las clases que los implementan.
  - El paquete Input representa las clases que permiten generar los datos de entrada a través de la interfaz gráfica, lectura de un archivo txt o xml, o generando los datos aleatoriamente.
  - El paquete Output representa las clases que muestran los resultados en la interfaz gráfica y también se almacenan en un archivo.

- El paquete GUI contiene la clase que contiene la implementación de la interfaz gráfica. Esta clase es común para todos los algoritmos y se adapta a cada problema.
- El paquete Main contiene la clase principal del proyecto.

La estructuración de las clases es la misma para el proyecto que abarca la aplicación de Android, excepto las que contienen la implementación de la interfaz gráfica. Para esta aplicación no se ha realizado la implementación que permite leer los datos de entrada a través de un archivo.

Esta organización del proyecto no requiere realizar muchas modificaciones al añadir un algoritmo nuevo o una variante nueva común para todos los algoritmos.

La profundización de la explicación del proyecto de Java (algoritmos, herramientas, implementación, etc.), tanto para escritorio como para Android, se realiza en los apartados de desarrollo de la aplicación e implementación de la aplicación.

5. Funcionalidades de la Aplicación

En este apartado explicamos brevemente las funcionalidades que tiene la aplicación:

- Se puede generar los datos de entrada de los problemas a resolver aleatoriamente, introduciendo los datos a través de la interfaz gráfica, o leyendo de un archivo (txt o xml) con un formato específico. Los datos generados a través de la interfaz gráfica también se pueden almacenar en un fichero de texto para pruebas futuras.
- Mostrar los atributos, productores y perfiles de clientes generados (también subperfiles en el caso del algoritmo genético).
- Editar y representar las entradas (instancias) de los dos problemas mencionados anteriormente.
  Los resultados obtenidos al ejecutar cualquier algoritmo se pueden visualizar en la interfaz gráfica y también se guardan en un fichero de texto.
- Estimar los precios de cada producto en función de sus características. Una característica relevante de cualquier producto, que afecta fuertemente a las preferencias de los clientes, es su precio, y este depende de las características del producto. En nuestro caso de estudio, necesitaremos estimar el precio de cada posible producto. Por ello, nuestra aplicación permitirá hacer tales estimaciones. Concretamente, la aplicación permite estimar una función con tantos parámetros de entrada como atributos que nos devuelva el precio. Cada producto posible (combinación de valores de atributos) es un punto de la función: dada sus características, su precio es lo que devolverá la función.
  La inclusión de características como el precio hará que el producto óptimo no necesariamente deba consistir en la combinación de las características más caras (frecuentemente las preferidas). Al encarecer el producto, lo hacen menos atractivo.
- Además de los problemas PDO, SPDG en sus versiones básicas, se resuelven variantes de los problemas propuestos, como que cada productor pueda poner en el mercado simultáneamente más de un producto, y que ciertas combinaciones de valores de atributos tengan bonificaciones o penalizaciones para ciertos perfiles de clientes si están presentes juntos. La aplicación permitirá visualizar y almacenar todos los resultados obtenidos por sus algoritmos en la resolución de los problemas considerados.
- En base a las respuestas obtenidas de las encuestas realizadas en Feebbo (para construir a partir de ellas las instancias de nuestros problemas a resolver en nuestro caso de estudio), podremos agrupar a los encuestados

en diferentes perfiles que tengan similitudes entre sí. La aplicación muestra la información sobre los clusters en los que se han agrupado y, a su vez, los clientes que pertenecen a cada uno. Los centroides obtenidos al realizar la clusterización representan nuevos perfiles, esto nos permite crear nuevos individuos de las poblaciones iniciales de nuestros algoritmos heurísticos para PDO que ayuden a estos a encontrar mejores soluciones.

Para el desarrollo de la aplicación hemos tomado las siguientes decisiones:

- Utilizaremos el lenguaje de programación Java y desarrollaremos una interfaz gráfica para escritorio y una aplicación para Android.
- Para estimar el precio de cada producto, aplicaremos un algoritmo de interpolación.
- Para realizar la clasificación de perfiles organizada en clusters utilizaremos un algoritmo de agrupamiento utilizando Weka en Java.
- Se han implementado algoritmos similares al algoritmo genético para comparar las eficiencias entre ellos: algoritmo de optimización por enjambre de partículas y algoritmo de enfriamiento simulado.
- Cada algoritmo podrá resolver todas las variantes consideradas para sus correspondientes problemas.

6. Desarrollo de la Aplicación

Tras explicar el diseño de nuestra aplicación, definiremos el desarrollo del mismo. Justificaremos la elección del lenguaje de programación escogido, y explicaremos los algoritmos y las herramientas que hemos decidido utilizar para el desarrollo del proyecto.

6.1 Java

Antes de iniciar el desarrollo de nuestra aplicación, tuvimos que decidir qué lenguaje de programación íbamos a utilizar. Decidimos que sería *Java*, ya que conocíamos ese lenguaje y además nos resultaría más práctico para la creación de la aplicación móvil. Además, *Java* tiene una variedad de librerías que podemos utilizar, lo que en nuestro caso fue una gran ventaja, pues hicimos uso de ellas.

Por otro lado, hemos tenido en cuenta la diferencia de eficiencia entre Java y Visual Basic (lenguaje utilizado en la versión inicial de los algoritmos). Este segundo lenguaje es un poco más rápido que Java, pero no dificulta al realizar las ejecuciones.

6.2 Algoritmo Genético

Los algoritmos genéticos [7] [8] son técnicas de inteligencia artificial que permiten encontrar soluciones para una gran variedad de problemas. Se basan en simular el procedimiento por el que la selección natural ha producido distintas especies de seres vivos: crear una “población” de soluciones diferentes, eliminar las peores, modificar y combinar entre sí las buenas y volver a empezar el ciclo aplicando la selección a estas soluciones modificadas.

Para ello, se parte de una población inicial, que puede ser generada aleatoriamente o no, de la cual se hace una selección de los individuos que están más capacitados para que luego se reproduzcan y se muten, y así finalmente obtener la siguiente generación de individuos que estarán más adaptados que la anterior.

Los tres tipos de operadores del Algoritmo Genético son:

- Selección o reproducción: su labor es escoger los cromosomas entre la población para llevar a cabo la reproducción. Cuanto más capaz sea el cromosoma, más veces será seleccionado para reproducirse.
- Cruce: su labor es crear los cromosomas de la descendencia mezclando los cromosomas de sus progenitores.
- Mutación: Este operador produce variaciones de modo aleatorio en un cromosoma. La mutación puede darse en cada posición de un bit en una cadena con una probabilidad, normalmente muy pequeña.

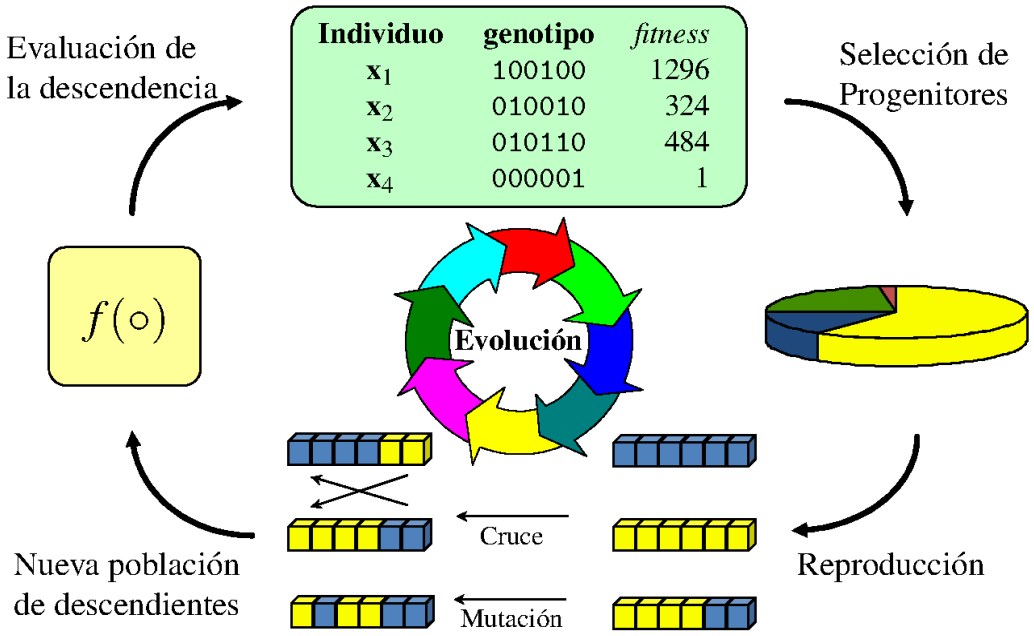


Figura 1 Diagrama Algoritmo Genético[1]

[1] Algoritmo Genético Clásico: http://escritura.proyectolatin.org/inteligencia-artificial/algoritmo-genetico-clasico/

```
BEGIN /* Algoritmo Genetico Simple */
    Generar una poblacion inicial.
    Computar la funcion de evaluacion de cada individuo.
    WHILE NOT Terminado DO
    BEGIN /* Producir nueva generacion */
        FOR Tamaño poblacion/2 DO
        BEGIN /*Ciclo Reproductivo */
            Seleccionar dos individuos de la anterior generacion,
            para el cruce (probabilidad de seleccion proporcional
            a la funcion de evaluacion del individuo).
            Cruzar con cierta probabilidad los dos
            individuos obteniendo dos descendientes.
            Mutar los dos descendientes con cierta probabilidad.
            Computar la funcion de evaluacion de los dos
            descendientes mutados.
            Insertar los dos descendientes mutados en la nueva generacion.
        END
        IF  la poblacion ha convergido THEN
            Terminado := TRUE
    END
END
```

Figura 2 Pseudocódigo del Algoritmo Genético[2]

En el problema de conseguir el mejor producto de forma automática, utilizamos este método para lanzar muchas soluciones e iterativamente quedarnos con las que mejor funcionan.

Teníamos dos formas de implementar el Algoritmo Genético:

- La primera opción era usar un framework de Java: JGAP (Java Genetic Algorithms Package) [9], y aplicarlo a nuestro proyecto.
- La segunda opción era partir de una implementación hecha por el profesor Pablo Rabanal para resolver el mismo problema (diseñar un producto que maximice el número de clientes o para llegar a un número determinado de clientes objetivo). En caso de escoger esta opción, tendríamos que adaptar dicha implementación al lenguaje que íbamos a utilizar, además de hacer los cambios convenientes.

Finalmente decidimos llevar a cabo la segunda opción, ya que la adaptación a nuestra aplicación sería más fácil.

[2] Algoritmos Genéticos: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf

6.3 Algoritmo de Optimización por Enjambre de Partículas

El algoritmo de Optimización por Enjambre de Partículas (en inglés Particle Swarm Optimization, de ahora en adelante utilizaremos sus siglas PSO) es una técnica de optimización en el campo del aprendizaje automático.

Hace referencia a una metaheurística inspirada en el comportamiento social de las bandadas de aves al volar, así como a los movimientos de los bancos de peces. Una población de entidades se mueve por el espacio de búsqueda durante la ejecución del algoritmo. Estas entidades son muy simples y realizan interacciones locales. El resultado de la combinación de comportamientos simples es la aparición de comportamientos complejos y la posibilidad de obtener buenos resultados en equipo.

Se parte del supuesto de que la bandada busca comida en un área (el objetivo del problema). Una buena estrategia para encontrar la comida es seguir al ave que más esté comiendo en su posición, lo que indica que en dicha posición hay más comida (mejor valor objetivo). PSO emula este comportamiento para resolver problemas de optimización. Cada solución o partícula es un ave en el espacio de búsqueda que está siempre en movimiento y nunca muere. De hecho, se puede considerar que el enjambre es un sistema multiagente donde las partículas son simples agentes que se mueven a través del espacio de búsqueda y memorizan la mejor solución encontrada hasta el momento.

Cada elemento del enjambre tiene un valor de fitness, una posición y un vector de velocidad que dirige su vuelo.

Las siguientes ecuaciones ajustan la velocidad y la posición de cada partícula:

$$V_i^{t+1} = w^t V_i^t + c_1 r_1^t (P_i - X_i^t) + c_2 r_2^t (G_i - X_i^t)$$

$$X_i^{t+1} = X_i^t + V_i^{t+1}$$

Donde:

V_i^{t+1} es la velocidad ajustada

w^t es la inercia del propio movimiento

c_1 es el coeficiente de confianza en la experiencia

c_2 es el coeficiente de confianza en la experiencia del grupo

P_i es la mejor posición previa de i

X_i^t es la posición actual de i

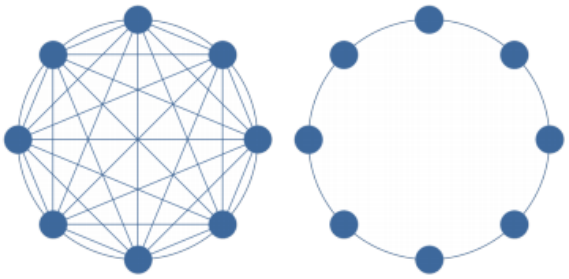
G_i es la mejor posición previa encontrada por el grupo

r_1^t y r_2^t son los operadores aleatorios entre 0 y 1

X_i^{t+1} es la posición de la partícula i después del ajuste

El movimiento de las partículas es guiado en parte por la mejor solución encontrada por todas las partículas. Concretamente, una partícula está compuesta de cuatro vectores:

1. Un vector que almacena la posición actual (localización) de la partícula en el espacio de búsqueda.
2. Un vector que almacena el vector de velocidad de acuerdo al cual la partícula se mueve.
3. Un vector que almacena la localización de la mejor solución encontrada por la partícula hasta el momento.
4. Un vector que almacena la localización de la mejor solución encontrada por el enjambre y es común para todas.



(a) Topología de estrella (gbest) (b) Topología de anillo (lbest)

Figura 3 Ejemplos de mejor solución global (*Gbest*) y local (*Lbest*)[3]

También hay tres valores de fitness:

1. El primer valor almacena el fitness de la solución actual.
2. El segundo valor almacena el fitness de la mejor solución local.
3. El tercer valor almacena el fitness de la mejor solución global.

[3] http://www.yorku.ca/sychen/research/theses/2011_Yenny_MSc.pdf

El movimiento de cada partícula depende de la mejor solución que ha encontrado desde que se inició el algoritmo y de la mejor solución encontrada por todas las partículas en toda la nube de partículas. Este tipo de vecindad de las partículas, donde todas las partículas en el enjambre son atraídas por la mejor solución, proporciona una rápida convergencia del algoritmo. De acuerdo con esta topología, la velocidad se cambia en cada iteración del algoritmo para acercarla a posiciones donde está la mejor solución local/global encontrada.

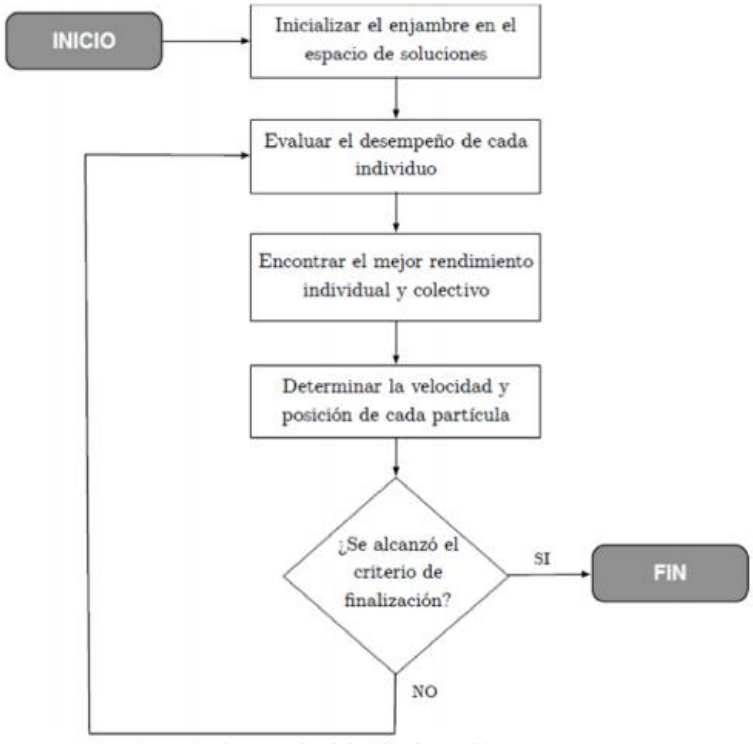


Figura 4 Esquema general de un algoritmo PSO[4]

6.4 Algoritmo de Enfriamiento Simulado

El algoritmo de enfriamiento simulado (en inglés Simulated Annealing, de ahora en adelante utilizaremos sus siglas SA) es un método para encontrar una solución buena (no necesariamente perfecta) a un problema de optimización.

En este algoritmo se mantiene una variable de temperatura para simular el proceso de enfriamiento. Mientras la variable de temperatura es alta, se permite al algoritmo, con mayor frecuencia, aceptar las soluciones que son peores que la solución

[4] http://blade1.uniquindio.edu.co/uniquindio/revistainvestigaciones/adjuntos/pdf/2bc6_46-53.pdf

actual. Esto le da al algoritmo la capacidad de saltar desde cualesquiera óptimos locales que se enfrenta al comienzo de la ejecución. A medida que se reduce la temperatura, también se reduce la posibilidad de aceptar soluciones peores, por lo que el algoritmo se concentra gradualmente en la zona del espacio de búsqueda se esperan encontrar soluciones cercanas a la solución óptima.

Este proceso gradual de 'enfriamiento' es lo que hace el algoritmo de enfriamiento simulado sea notablemente eficaz en la búsqueda de una solución cercana a la óptima cuando se trata de problemas que contienen numerosos óptimos locales. En comparación a otros algoritmos como el de Hill Climbing, nos permite evitar el problema de quedarnos atascados en óptimos locales, y es mucho mejor en promedio en su aproximación al óptimo local.

Para que el algoritmo decida qué nuevas soluciones aceptar para evitar los óptimos locales, comprueba si la solución vecina considerada en cada momento es mejor que la que tiene actualmente. Si es así, se acepta incondicionalmente. Sin embargo, si esa solución no es mejor, se tienen en cuenta más factores: cuánto peor es la solución vecina; y la temperatura actual. A altas temperaturas es más probable que acepte soluciones que son peores.

La ecuación para calcular la probabilidad de aceptación es la siguiente:

$$a = e^{\Delta E/T}$$

Donde a es la probabilidad de aceptación, E es la energía del sistema y Δ es la diferencia entre el estado actual y el nuevo valor calculado, T es la temperatura y e es 2.71828.

Básicamente, cuanto menor sea la diferencia de costes y más alta la temperatura, más probable es que el algoritmo acepte la nueva solución.

6.5 Algoritmo Minimax

El Algoritmo Minimax [10] consiste en utilizar una función de evaluación heurística exhaustivamente para cierto número de configuraciones futuras posibles de un juego. Es un algoritmo de decisión que ayuda a minimizar la máxima pérdida esperada en un juego de adversarios.

El algoritmo trata de seleccionar el mejor movimiento suponiendo que nuestro contrincante elegirá después el peor para nosotros. Esto se realiza de forma recursiva hasta que se cumpla alguna de las siguientes condiciones:

- Que gane algún jugador.
- Que ya se hayan explorado N niveles, siendo el N el número máximo de niveles (turnos futuros) a explorar.
- Que se haya terminado el tiempo de exploración.
- Que se haya llegado a un nivel donde no se puede realizar ningún cambio.

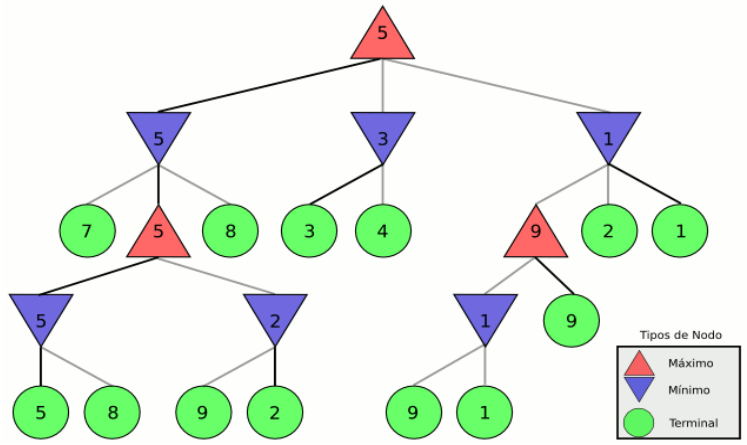


Figura 5 Ejemplo de Algoritmo Minimax[5]

La aplicación a nuestro problema se explicará en más detalle posteriormente, en la sección de implementación.

6.6 Algoritmo de Interpolación

Para definir las instancias a abordar en nuestro caso de estudio, necesitamos estimar el precio de cada posible teléfono móvil en función de sus características, pues por cada característica añadida, el precio del móvil aumenta, pero si tiene menos características (o más baratas), el precio disminuye. Para calcular la estimación del precio utilizamos métodos de interpolación.

El problema de la interpolación [11] consiste en, partiendo de una función de la que sólo conocemos una serie de puntos, hallar el valor de nuevos puntos de esa función, o incluso obtener una aproximación de la expresión que define la propia función.

[5] Wikipedia - Ejemplo de Algoritmo Minimax: https://es.wikipedia.org/wiki/Minimax

En nuestro caso, necesitamos realizar una interpolación multivariable [12], ya que tenemos que tener en cuenta más de una característica de un teléfono móvil.

Investigando en el campo, encontramos algunas herramientas que a priori podrían ser válidas: ArcGIS [13] y MatLab [14].

ArcGIS consiste en un conjunto de herramientas que tienen diferentes funcionalidades, entre ellas, modelado y análisis espacial en información ráster. Lo que más nos interesaba de todas sus herramientas era la de Interpolación [15], pero pronto nos dimos cuenta que los valores de entrada que utilizaba ArcGIS eran necesariamente imágenes. Por lo tanto, tuvimos que descartar esta herramienta.

MatLab nos proporcionaba diferentes funciones para interpolar datos con los distintos algoritmos existentes [16], pero también decidimos no utilizarlo porque necesitábamos probar interpolación con más de 50 variables, opción que no proporcionaba directamente MatLab.

Así que decidimos implementar nuestro propio método de interpolación. Teníamos dos formas de hacerlo:

1. Estimar una función *lineal* que tenga tantos parámetros de entrada como características tenga el móvil y nos devuelva el precio. Se puede interpolar a partir de los precios de móviles reales que utilizaremos para modelizar la competencia, de forma que cada uno de estos móviles (junto con su precio conocido) es un punto de dicha función. Este modelo lineal no tiene en cuenta que el precio de una característica puede depender de que ciertas características vayan juntas o no. Para estimar heurísticamente los pesos de cada parámetro en la función lineal, podemos utilizar un algoritmo genético: cada individuo (cromosoma) es una combinación concreta de pesos, y la función de fitness devuelve la suma de las distancias entre el precio real de cada móvil que se conoce y el precio que estimaría esa función (o el cuadrado de esas distancias, para que alejarse por mucho penalice más).
2. El valor de un punto X de una función se calcula como la media ponderada del valor observado de dicha función en todos los puntos (X1, X2, ..., Xn) de la muestra conocida, donde el peso de cada Xi es inversamente proporcional a su distancia a X.

De estas dos opciones decidimos hacer la implementación de la media ponderada, utilizando la siguiente fórmula:

$$precio(x) = \sum_{i \in producto} (precio(xi) * \frac{distancia(x, xi)}{\sum_{j \in producto} distancia(x, xj)})$$

donde asumimos que $distancia(a, b)$ es la distancia que hay desde el producto *a* hasta el producto *b*, definidos por los valores de sus atributos ($a_1, \dots, a_n$ y $b_1, \dots, b_n$ respectivamente).

$$distancia(a, b) = \sqrt{|a_1 - b_1|^2 + |a_2 - b_2|^2 + |a_3 - b_3|^2 + \cdots + |a_n - b_n|^2}$$

6.7 Algoritmo de Agrupamiento

Hemos añadido el estudio de este tipo de algoritmos al desarrollo de nuestro proyecto, debido a que era el más idóneo para la colaboración con la empresa Feebbo, ya que necesitaban un programa que les permita clasificar a sus clientes en grupos organizados por similitud. El trato que hicimos con Feebbo fue información por información, es decir, ayudarles con clusterización a cambio de usar su plataforma para poder realizar nuestra encuesta.

Además, nos pareció interesante incorporarlo en nuestra aplicación para clasificar los usuarios de móviles. Concretamente, podemos crear la población inicial del algoritmo genético, PSO, etc., de forma que, para cada clúster de perfiles de clientes, uno de los individuos iniciales sea un producto diseñado (vorazmente) para resultar atractivo a los perfiles agrupados en dicho clúster.

Un algoritmo de agrupamiento [17] [18] (llamado también clustering) es un procedimiento de agrupación que consiste en dividir las instancias de un conjunto de datos por grupos de instancias similares. Para medir la similitud entre objetos se suelen utilizar diferentes formas de distancia: distancia Euclídea, de Manhatan, etc.

Los métodos de agrupamiento se dividen en tres grupos fundamentales: jerárquicos, particionales y basados en densidad. En este caso, hemos decidido utilizar un algoritmo que pertenece al grupo de los particionales, es decir, que son aquellos que realizan una división inicial de los datos en grupos y luego mueven los objetos de un grupo a otro según se optimice alguna función objetivo.

Estos algoritmos asumen un conocimiento a priori del número de clusters en que debe ser dividido el conjunto de datos, y proporcionan una división en clases que optimiza un criterio predefinido o función objetivo. El algoritmo que vamos a emplear es *K-means.*

6.7.1 K-MEANS

K-means es uno de los métodos de agrupamiento más conocidos y más utilizados en aplicaciones científicas e industriales [17].

La idea principal es escoger k centroides al azar que representarán el centro del grupo que pretender representar, luego tomar cada punto del conjunto de datos y situarlo en la clase de su centroide más cercano. El siguiente paso es recalcular el centroide de cada grupo y volver a distribuir todos los objetos según el centroide más cercano. Este proceso es iterativo, ya que se repite hasta alcanzar el criterio de parada, por ejemplo, que no haya ningún cambio en los grupos de un paso al siguiente.

A diferencia de otros algoritmos, k-medias necesita la previa especificación del número de clusters que se desean obtener.

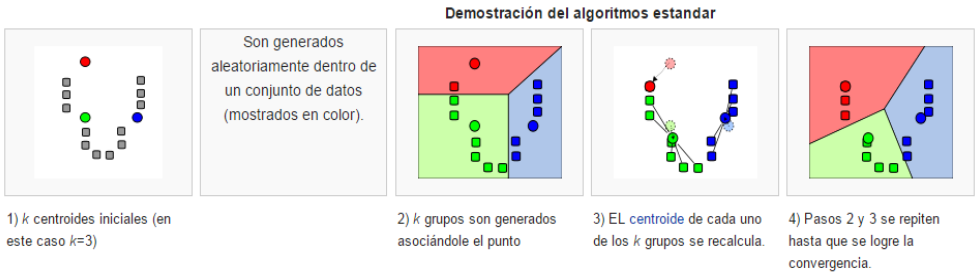


Figura 6 Diagrama Algoritmo K-MEANS[6]

Realizamos una investigación para ver si existía alguna herramienta de aprendizaje automático que nos permita utilizar este tipo de algoritmo y si nos sería útil, o por el contrario tendríamos que realizar nosotros la implementación. Finalmente encontramos la herramienta Weka, que nos fue fácil de usar siguiendo un tutorial básico [19] [20], así como el api [21] de esta herramienta.

La implementación del algoritmo explicado en este apartado es también la que tiene Weka.

6.7.2 WEKA

Weka [22] es una plataforma de software que contiene una colección de algoritmos de aprendizaje automático para tareas de minería de datos, escrito en Java y desarrollado en la Universidad de Waikato. Es un software de código abierto con una licencia GNU-GPL [23].

[6] Algoritmo K-Means: https://es.wikipedia.org/wiki/K-means

Los algoritmos se pueden aplicar a un conjunto de datos por medio de su interfaz o bien pueden ser llamados directamente desde su propio código Java [24].

Weka contiene diversas herramientas de datos: pre-procesamiento, clasificación, regresión, clustering, reglas de asociación, y la visualización.

Además, esta herramienta cuenta con un repositorio de ejemplos [25], con un tipo de formato específico que podemos utilizar para realizar distintas pruebas y ver cómo funciona.

Figura 7 Herramienta WEKA

En nuestro caso, utilizamos la opción clustering, y aplicamos el algoritmo SimpleKMeans. Tenemos la opción de elegir el tipo de distancia. Nosotros seleccionamos distancia Euclidean. Además, podemos decidir el número de clusters que queremos que obtener.

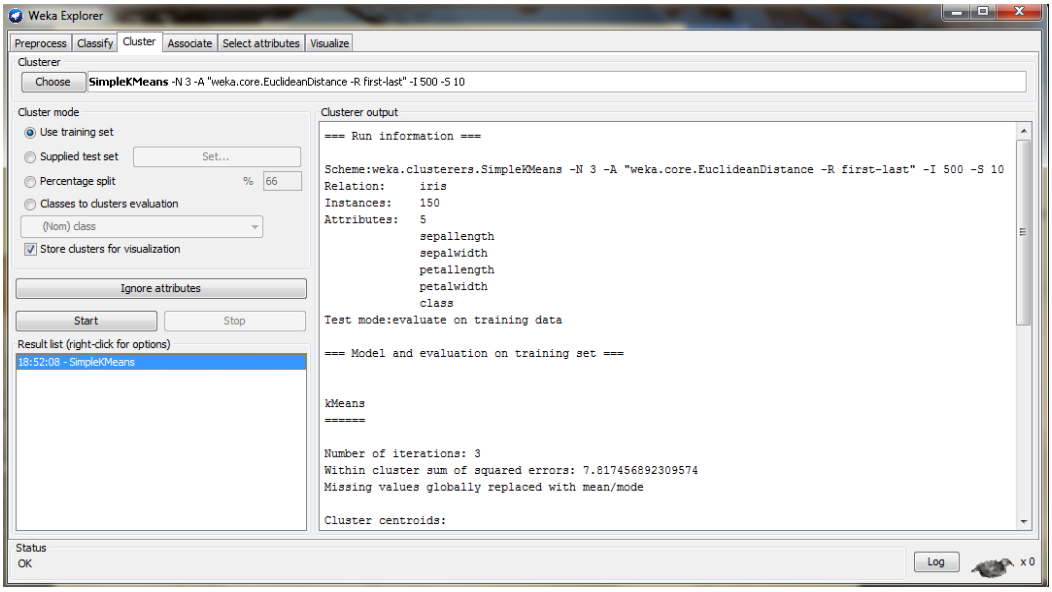


Figura 8 Herramienta WEKA – Algoritmo SimpleKMeans

En la colaboración con la empresa Feebbo para clasificar sus clientes en clusters por similitud, decidimos utilizar la herramienta Weka, puesto que nos permitía visualizar los resultados y obtenerlos de forma automática.

Por otro lado, también añadimos a nuestra implementación de Java la librería de Weka, para integrarlo en nuestra aplicación como nueva funcionalidad y poder clasificar a los usuarios representados en cualquier instancia con la que deba trabajar nuestra aplicación.

7 Implementación de la aplicación

A continuación, detallaremos más en profundidad cómo hemos implementado nuestra aplicación. Describiremos que librerías nos han hecho falta y las implementaciones de los algoritmos: genético, minimax, PSO y SA. Además, se explicará la aplicación para escritorio y para Android.

7.1 Librerías Utilizadas

7.1.1 Weka.jar

Es una librería de Java que permite realizar las mismas funcionalidades de la herramienta Weka. Su descarga es gratuita bajo las condiciones de una licencia GNU-GPL, que se están explicadas de forma detallada en su página web [22].

7.1.2 JavaCSV.jar

Es una librería de Java que permite leer y escribir archivos CSV bajo una licencia de ASL. Utilizamos esta librería para crear un archivo CSV con todos los perfiles de clientes generados para luego poder convertirlo al formato arff de Weka para realizar la clusterización.

7.2 Product Design

Nuestro proyecto de Java *ProductDesign* almacena la implementación de los siguientes algoritmos: genético, minimax, PSO y SA. Estos algoritmos a su vez resuelven diversas variantes de nuestros problemas: maximizar ingresos en lugar de clientes (usando la interpolación para fijar los precios), permitir a cada productor vender simultáneamente más de un producto, o dar bonificaciones o penalizaciones de valoración a combinaciones de valores de atributos que vayan juntas. Además, podemos utilizar la clusterización para crear individuos iniciales del génetico y el PSO a partir de los clusters de clientes o no.

7.2.1 Paquete General

Este paquete almacena clases comunes para todos los algoritmos: *Attribute*, *CustomersProfiles, Producer, Product, LinkedAttribute, Interpolation, Algorithm*.

7.2.1.1 Attribute

La clase *Attribute* representa la estructura de cada uno de los atributos del producto a generar. Cada atributo tiene un número mínimo y máximo de valores.

Además, cada uno tiene una lista de valores disponibles, que indican cuáles de esos valores están disponibles para ser producidos por el correspondiente productor.

El atributo también tiene una lista scoreValues, la cual tiene los datos de la valoración que le dan los perfiles de usuario a cada valor posible.

Dispone de getters y setters comunes en las demás clases.

7.2.1.2 Customer Profile

La clase *CustomerProfile* representa la estructura de un perfil de usuario. Esta estructura contiene datos como el número de personas en este perfil. La suma de este valor para todos los perfiles dará como resultado el número total de usuarios sobre el que se realiza el estudio.

También dispone de una lista de atributos, los cuales tendrán, en el campo de scoreValues mencionado en la clase Attribute, la valoración que le da este perfil a esos valores. Disponemos también de una lista de los subperfiles de usuario, obtenidos a partir de este perfil.

Además, tiene una lista de atributos enlazados para la variante que permite que cada par de atributos produzcan una bonificación o penalización de valoración.

Dispone de un constructor, métodos getters y setters comunes en las demás clases.

7.2.1.3 Producer

La clase *Producer* representa cada productor, es decir, a cada uno de los competidores, incluido el productor cuyo producto queremos optimizar.

Cada productor tiene un nombre, una lista de atributos disponibles y una lista de productos.

Dispone de un constructor, métodos getters y setters comunes en las demás clases.

7.2.1.4 Product

La clase *Product* representa cada individuo de la población. Cada individuo tiene una lista de atributos con sus valores correspondientes y una variable que almacena el precio del producto.

Implementa la interfaz cloneable, concretamente el método clone, que a partir un producto devuelve otro nuevo.

Dispone de un constructor, métodos getters y setters comunes en las demás clases.

7.2.1.5 Linked Attribute

La clase *LinkedAttribute* almacena dos o más atributos que pueden ir enlazados y sus valores correspondientes, junto con una variable que almacena la nueva puntuación (bonificación o penalización por ir juntos).

Dispone de un constructor, métodos getters y setters comunes en las demás clases.

7.2.1.6 Interpolation

La clase *Interpolation* representa la estimación del precio de un producto a partir del cálculo de la media ponderada explicada en el apartado de algoritmo de interpolación.

7.2.1.7 Problem

La clase *Problem* define la implementación de nuestro problema en particular.

En esta clase definimos el método *statistics* el cual nos ayuda a guardar la información para ofrecer los resultados finales. En esta clase ofrecemos los métodos necesarios para que los distintos algoritmos tengan las herramientas suficientes para generar sus espacios iniciales.

En esta clase quedan a la disposición de los algoritmos métodos que obtienen el espacio de soluciones del problema a solucionar, para que quede totalmente abstraído el problema de los algoritmos, así el algoritmo no tiene que ser específico para nuestro problema, y con que se le ofrezcan este tipo de funciones ya sería capaz de solucionarlo.

El método *getFitnesss* es implementado en esta clase, y con él los métodos para la puntuación de un producto y de sus atributos, y las funciones de *computeWSC* para calcular los clientes acumulados, y *computeBenefits* para calcular ganancias en lugar de clientes (ya que están funciones son iguales para todos los algoritmos).

Para poder calcular los beneficios también se incluyen en esta clase los métodos para la interpolación y los precios de la competencia, para poder obtener el precio del propio producto.

Por último, tenemos el método *solveProblem*, el cual implementa cada uno de los algoritmos para solucionar el problema cada uno a su manera. De esta manera, el problema se ve independiente de los algoritmos, y de lo único que se tiene que preocupar el de llamar al método *solveProblem*, y esperar el resultado óptimo del problema, independiente del algoritmo usado.

7.2.2 Paquete Genetic

Nuestra resolución del problema PDO utilizando el Algoritmo Genético está basada en la implementación previa que realizó el profesor Pablo Rabanal antes de comenzar este proyecto. Nuestra versión mejorada de dicho algoritmo se ha construido de tal manera que se mantiene abstraído el algoritmo genético en sí, como parte independiente del problema, y por separado, nuestra implementación del problema concreto a resolver.

7.2.2.1 Genetic Algorithm

La clase *GeneticAlgorithm* representa la implementación del algoritmo genético independiente del problema que se le pase.

En esta clase definimos:

- La población, que representa el conjunto de individuos que estamos intentando optimizar.
- Una lista que representa el valor de la función de Fitness para cada uno de los individuos de la población.
- Una variable para guardar el mejor individuo obtenido hasta el momento.
- Variables globales como el número de generaciones, el número de individuos de la población y la probabilidad de mutación y de cruce entre dos individuos.

Los métodos abstractos a implementar por el problema son los siguientes:

- Método para proveer al algoritmo la población inicial a partir de la cual tiene que comenzar.
- La función de mutación, que define de qué manera es posible mutar a cada individuo.
- La función de cruce, que define cómo se juntan dos individuos para dar paso al siguiente individuo de la generación.
- Y por último la función de fitness, que contiene la heurística para saber qué individuo es mejor que otro, y así poder evolucionar la población.

El método inicial de esta clase comienza con una población inicial obtenida a través del método abstracto mencionado antes. De ella obtiene su fitness y el mejor individuo de la población. A partir de aquí, el algoritmo comienza a iterar un número de generaciones, y para cada generación, realiza dos acciones.

Primero crea una nueva población a partir de la que tenemos. Para ello, itera tantas veces como individuos haya en la población y va eligiendo un padre y una madre. A partir de ellos genera un nuevo individuo, que después sufre una posible mutación. Así acabamos creando nuestra nueva población.

Una vez tenemos esto, utilizamos el método Tournament, para que todos los individuos de ambas poblaciones se enfrenten por parejas formadas aleatoriamente, comparando sus valores de Fitness. Así obtenemos una nueva población que contiene no solo a los mejores individuos de las dos generaciones, y así los peores son descartados. Además, después de obtener la nueva y mejorada población,

comprobamos si hemos conseguido un nuevo mejor individuo, y si es así lo actualizamos.

Cuando acaba el algoritmo, devolvemos la cadena de cromosomas del mejor individuo alcanzado.

7.2.2.2 SubProfile

La clase *SubProfile* representa cada uno de los subperfiles creados a partir de los perfiles de personas.

Cada subperfil tiene un nombre y una lista de atributos en la que elige un valor para cada uno de los ellos.

Dispone de un constructor, métodos getters y setters comunes en las demás clases.

7.2.3 Paquete Minimax

Nuestra resolución del problema SPDG y lSPDG utilizando el Algoritmo Minimax está basada inicialmente en la implementación que realizó el profesor Pablo Rabanal antes de comenzar este proyecto. Nuestra versión mejorada de dicho algoritmo se construyó en base a las siguientes clases principales.

7.2.3.1 MinimaxAlgorithm

La clase *MinimaxAlgorithm* representa la implementación del algoritmo minimax independiente del problema sobre el que se le quiera usar.

En esta clase definimos:

- Listas para almacenar los resultados acumulados hasta la fecha por cada jugador.
- Variables globales como la profundidad máxima y mínima del minimax, número de turnos y número de jugadores (que en nuestro caso es 2 siempre), y número de turnos previos sobre los que calcular los resultados.

Los métodos abstractos a implementar por el problema son los siguientes:

- Un método para obtener el Fitness.
- Métodos para obtener el espacio de movimientos de juego sobre el que trabajamos para poder realizar los cambios en el objeto en cada turno de juego.
- Un método (*setSolution*) para poder realizar el movimiento escogido por el jugador.

El método inicial de esta clase itera, para cada uno de los jugadores, todos los turnos hasta terminar el juego.

En cada uno de estos turnos se ejecutan dos métodos. El primero se encarga de cambiar el objeto a mejorar. Para ello, comienza a crear el árbol de soluciones, que mediante una función recursiva comienza a buscar el cambio óptimo.

Una vez hecho el cambio, se actualizan los resultados obtenidos por ese jugador, y se realiza el cambio mediante la función *setSolution*. A continuación, hacemos lo mismo con el otro jugador, y luego el turno vuelve a comenzar.

Al terminar el proceso, el objeto ideal debe quedar almacenado para cada jugador mediante la función setSolution que se ejecuta al final de cada turno.

7.2.3.2 StrAB

La clase *StrAB* representa la estructura de alfa-beta, la cual se usa para guardar los cambios obtenidos por las ramas del minimax, y poder compararlos al final del algoritmo.

7.2.4 Paquete PSO

En este paquete se encuentra la clase que representa la implementación del algoritmo de optimización por enjambre de partículas.

7.2.4.1 PSO Algorithm

La clase *PSOAlgorithm* representa la implementación del algoritmo PSO, independiente del problema al que quiera ser aplicado.

En esta clase definimos:

- Una lista que guarda el enjambre con el que realizaremos el algoritmo.
- Variables de configuración, como la velocidad tope, el factor de confianza en la experiencia, o valores para el cálculo de valoración de la propia inercia.
- La lista de las velocidades de las partículas del enjambre.
- Variables para guardar el Fitness obtenido de cada una de las partículas, así como la mejor partícula obtenida hasta la fecha.

Los métodos abstractos a implementar por el problema son los siguientes:

- Un método para obtener el Fitness.
- Un método para inicializar el enjambre inicial.
- Un método para obtener la localización actual de una partícula.
- Y un último método para actualizar la posición de una partícula.

El método inicial de esta clase (*solvePSOAlgorithm*) es el encargado comenzar con el algoritmo.

En primera instancia, creamos el enjambre inicial, y obtenemos para cada una de las partículas una velocidad aleatoria.

A continuación, iteramos por un número de veces determinado el enjambre, el cual se irá moviendo sobre nuestro espacio de soluciones para obtener la partícula óptima. En cada una de estas iteraciones habrá varios pasos.

El primer paso es actualizar nuestro enjambre con las posibles mejores partículas que haya encontrado. Inmediatamente después actualizamos el mejor individuo que tenemos, en el caso de haberlo mejorado.

En el último paso iteramos por cada partícula del enjambre, y para cada una de ellas, obtenemos una nueva velocidad, basada en su estado previo y en la posición de la mejor partícula encontrada. Después de tener la nueva velocidad, actualizamos la posición en base a ella.

Al terminar las iteraciones se actualizan los Fitness.

Cuando el algoritmo concluye, es devuelta la mejor partícula hallada.

7.2.5 Paquete SA

En este paquete se encuentra la clase que representa la implementación del algoritmo de enfriamiento simulado.

7.2.5.1 SA Algorithm

La clase *SimmulatedAnnealingAlgorithm* representa la implementación del algoritmo SA, independiente del problema al que quiera ser aplicado.

En esta clase definimos:

- La temperatura inicial.
- La temperatura actual.
- El ratio de enfriamiento.

Los métodos abstractos a implementar por el problema son los siguientes:

- Un método para obtener el Fitness.
- Un método para permitir que el algoritmo pueda cambiar el objeto con el que está trabajando.

El método inicial de esta clase (*solve_SA*) es el encargado comenzar con el algoritmo.

Trabajaremos con dos objetos, el mejor objeto encontrado, y el actual.

Este algoritmo comienza con la temperatura en su máximo, e itera hasta que la temperatura llega al mínimo. Durante la ejecución de cada iteración creamos un objeto nuevo, y luego obtenemos su Fitness mediante el método abstracto *getFitness*, comparamos los Fitness de cada producto y nos quedamos con el mejor o no, dependiendo de una probabilidad dependiente de la temperatura.

Cuando el algoritmo termina, el método principal devuelve el objeto óptimo, modelado a base de iteraciones.

7.2.6 Paquete Input

En este paquete se almacenan las clases que describen las distintas formas de leer los datos de entrada.

7.2.6.1 Input Random

La clase *InputRandom* representa los métodos que permiten generar los datos de entrada aleatoriamente. Genera atributos, productores, perfiles, subperfiles y números de perfiles de clientes. Los subperfiles se generaron para tener en cuenta

también los grupos minoritarios y no despreciar ninguna valoración. La creación de los subperfiles se omitió para el algoritmo Minimax, ya que es un algoritmo bastante costoso, y multiplicar el número de perfiles con los que juega lo haría más costoso todavía. También se pueden generar aleatoriamente los atributos enlazados (es decir, combinaciones de ellos con bonificaciones o penalizaciones si van juntos).

Dispone de un constructor, métodos getters y setters comunes en las demás clases.

7.2.6.2 Input GUI

La clase *InputGUI* representa los métodos que permiten leer los datos de entrada introducidos por la interfaz gráfica para generar atributos, productores, perfiles y números de perfiles de clientes. También se pueden generar los atributos enlazados introduciendo los datos correspondientes.

A su vez, los datos que se añaden a través de la interfaz gráfica pueden ser guardados en un archivo txt o xml para futuras pruebas. Además, en esta clase se encuentran también los métodos que permiten obtener los datos de entrada de un archivo txt y xml.

Para realizar la lectura y escritura de un archivo txt, utilizamos las clases de Java *Scanner* y *PrintWriter* respectivamente. Para realizar la lectura y escritura de un archivo xml utilizamos *DOM* (Document Object Model), que es un api de Java para el procesamiento de XML.

Dispone de un constructor, métodos getters y setters comunes en las demás clases.

7.2.6.3 Input Weka

La clase *InputWeka* representa la implementación del algoritmo de agrupamiento partiendo de los perfiles de clientes generados como datos de entrada. Los métodos utilizados en esta clase son importados de la librería de Weka en Java.

Creamos y configuramos el algoritmo SimpleKMeans para realizar la clusterización.

Cargamos los datos de entrada de un archivo csv y realizamos un filtrado que nos permite cambiar los atributos numéricos a nominales, con la finalidad no de

obtener números reales en los centroides, sino de que pueda seleccionar valores que están dentro de un rango específico.

Construimos el algoritmo de clusterización e identificamos el cluster de cada instancia, y por último calculamos los centroides.

Dispone de un constructor, métodos getters y setters que nos permiten obtener el número de clusters y modificarlo.

7.2.7 Paquete Output

En este paquete se almacenan las clases en los que se almacena los datos de salida.

7.2.7.1 Output CSV

La clase *OutputCSV* tiene un método que genera un archivo csv a través de los perfiles de clientes generados, con la finalidad de que ese archivo se pueda pasar como datos de entrada en Weka.

7.2.7.2 Output Results

La clase *OutputResults* tiene un método que genera un archivo txt que almacena los resultados de las ejecuciones de cada uno de los algoritmos.

7.2.7.3 Show Results

La clase *ShowResults* representa los métodos que muestran las listas generadas por cada algoritmo: atributos, productores, perfiles y subperfiles.

7.2.8 Paquete GUI

En este paquete se almacena la clase que representa la implementación de la interfaz gráfica para escritorio.

7.2.8.1 GUI

La clase *GUI* representa la interfaz gráfica del programa para visualizar los datos obtenidos por cada uno de los algoritmos, así como la implementación de los eventos con los que interactuamos.

Se realizan invocaciones a métodos las clases que representan cada algoritmo para solicitar su ejecución.

La interfaz muestra los resultados del algoritmo y el tiempo que tarda en ejecutarse. Además, permite realizar modificaciones de los datos de entrada y de datos generales, y también de los resultados que se obtienen al realizar la clusterización.

7.2.9 Paquete Main

7.2.9.1 Main

La clase *Main* representa la clase principal del programa. En esta clase llamamos a la clase GUI para ejecutar la aplicación.

7.3 ObjectAid UML Explorer

ObjectAid UML Explorer es una herramienta para organizar la creación y visualización de diagramas en UML en Eclipse.

En estos diagramas de clase UML se muestra los métodos del código fuente de Java. Cualquier cambio que se realice en el código de la clase Java automáticamente se ve reflejado en el diagrama.

En este caso, hemos elegido esta herramienta de visualización porque es muy sencilla de utilizar, y nos permite instalarlo como un plugin de Eclipse.

7.3.1 Diagrama de Clases

En este apartado mostramos un diagrama de clases con las clases más relevantes del proyecto Marketing Computacional.

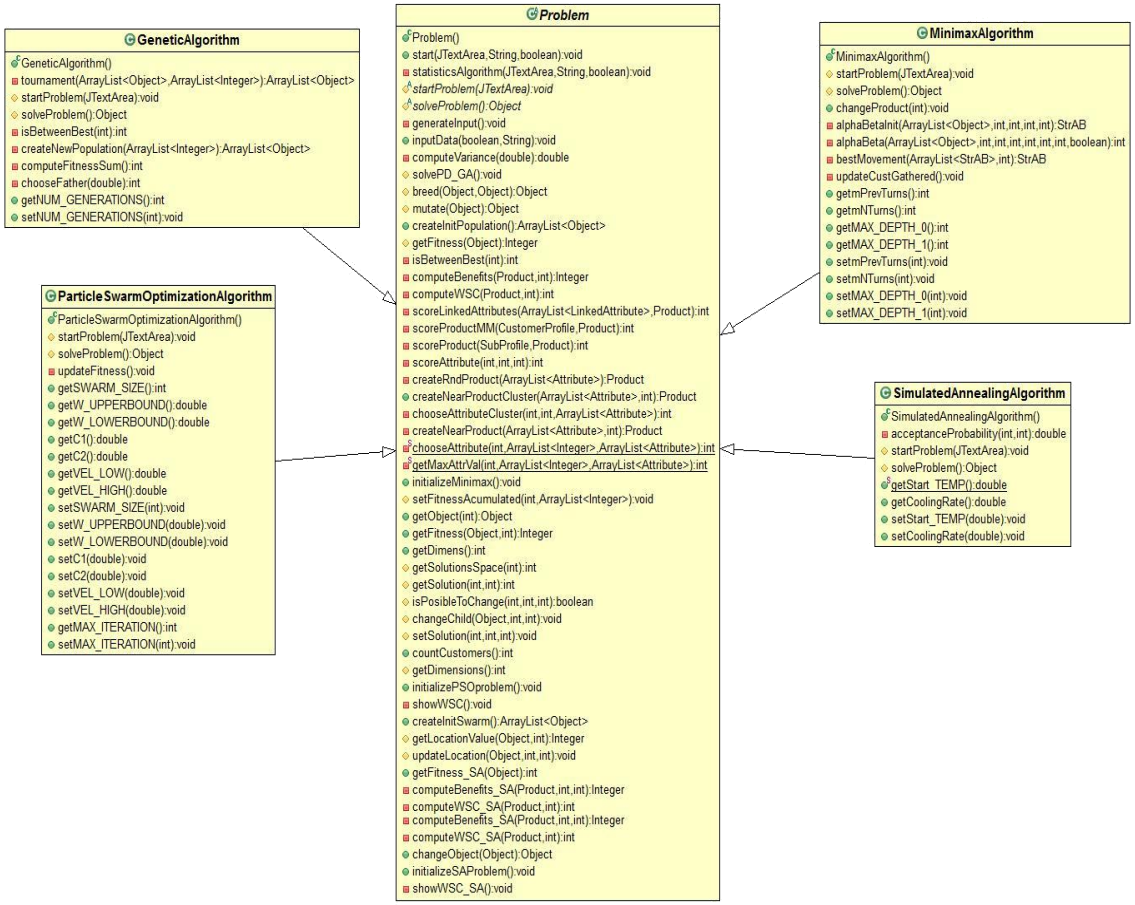


Figura 9 Diagrama de Clases Abstracción

En la Figura 9 se muestra el diagrama de clases que contiene la abstracción del código. La clase *Problem* contiene los métodos implementados para nuestro problema en particular. Las demás clases representan la implementación de los algoritmos y sus métodos correspondientes. Cada algoritmo tiene una relación directa con la clase *Problem.*

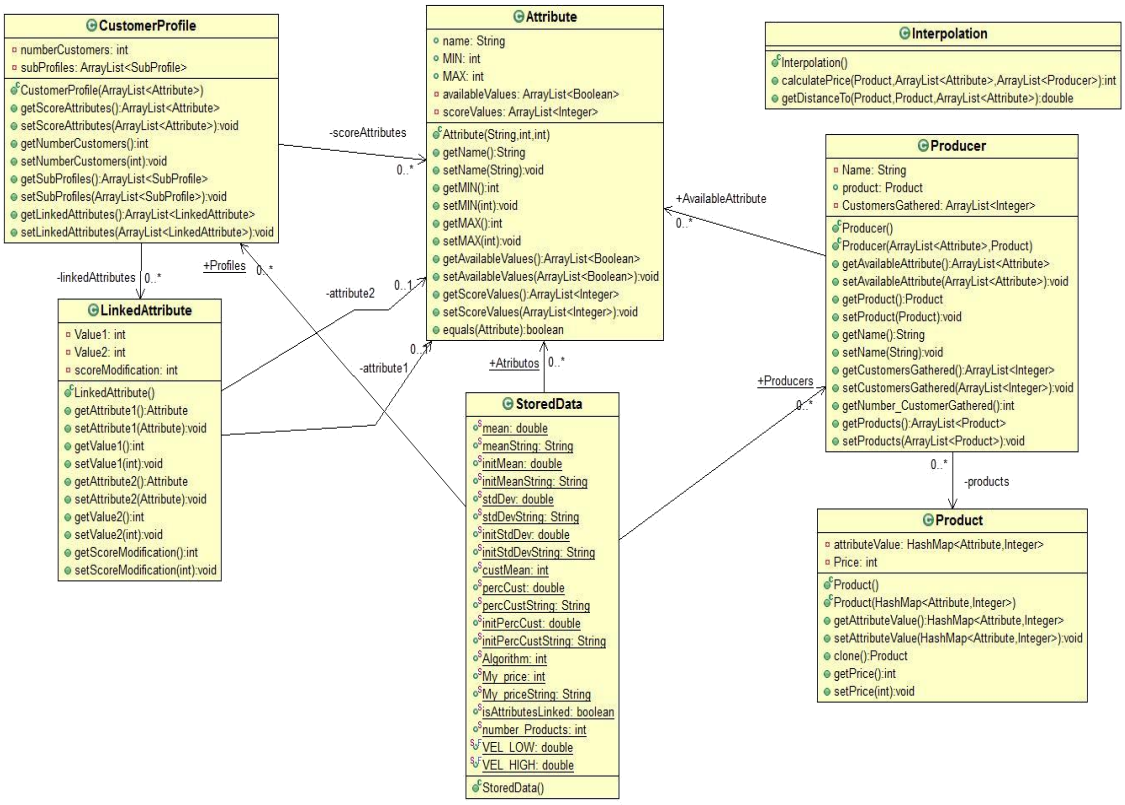


Figura 10 Diagrama de Clases Generales

En la Figura 10 se muestran las clases comunes que tienen todos los algoritmos que se han implementado en este proyecto y las relaciones directas que tienen entre sí.

7.4 Interfaz Gráfica

Para desarrollar la aplicación para escritorio, primero utilizamos NetBeans, que es un entorno de desarrollo integrado libre y gratuito que nos permite generar una interfaz de usuario. No obstante, nos dimos cuenta de que generaba muchas líneas de código y de que empeoraba el rendimiento. Finalmente, decidimos implementar la interfaz desde cero en Java.

Además, queríamos ofrecer al usuario la posibilidad de trabajar con nuestra aplicación de la forma que más cómoda le resulte. Por ello, decidimos hacer nuestra aplicación multiplataforma y, en particular, ofrecerla también para dispositivos Android.

Utilizamos Android Studio, que es un entorno de desarrollo integrado para la plataforma Android. La implementación de esta interfaz se realiza en Java, y además tiene una licencia Apache 2.0 de uso gratuito.

7.4.1 Interfaz para Escritorio

Nuestra interfaz consta de los siguientes paneles:

a) Un panel por cada algoritmo (genético, minimax, PSO y SA) que ofrece los resultados tras su ejecución.
b) Un panel que muestra las listas de atributos, productores, perfiles y subperfiles que se generan al realizar cada algoritmo.
c) Un panel para modificar los datos de entrada comunes para cada algoritmo, es decir, se pueden modificar los atributos, productores y perfiles. Al lado podemos ver qué atributos se están añadiendo con sus respectivas características.
d) Un panel que permite modificar los datos generales, tales como número de ejecuciones y porcentajes de atributos conocidos, etc. También se pueden modificar las variables propias de cada algoritmo.
e) Por último, un panel que muestra los resultados de Weka al realizar la clusterización. Podemos observar las instancias que pertenecen a cada cluster y cuáles son los centroides.

Mostramos unas imágenes de ejemplo de la interfaz gráfica para escritorio:

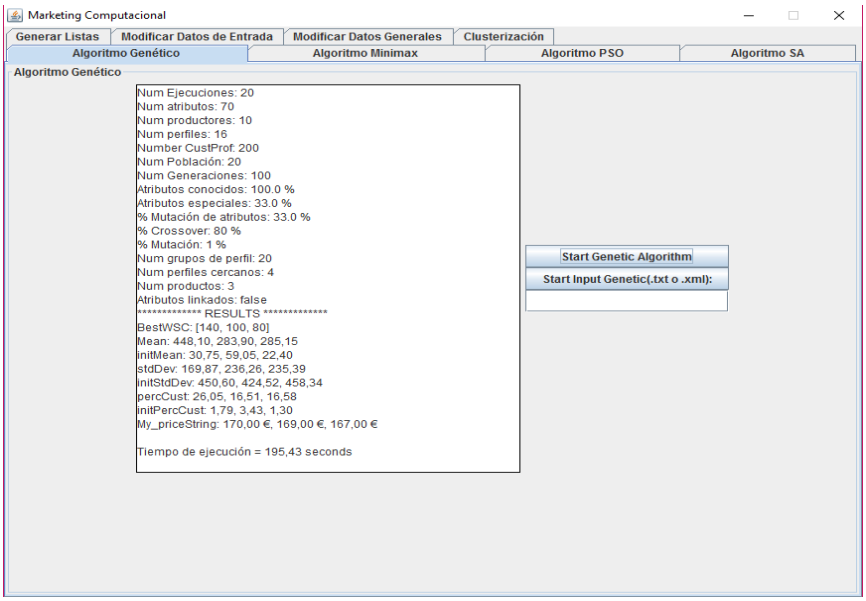


Figura 11 Ejemplo de ejecución Algoritmo Genético

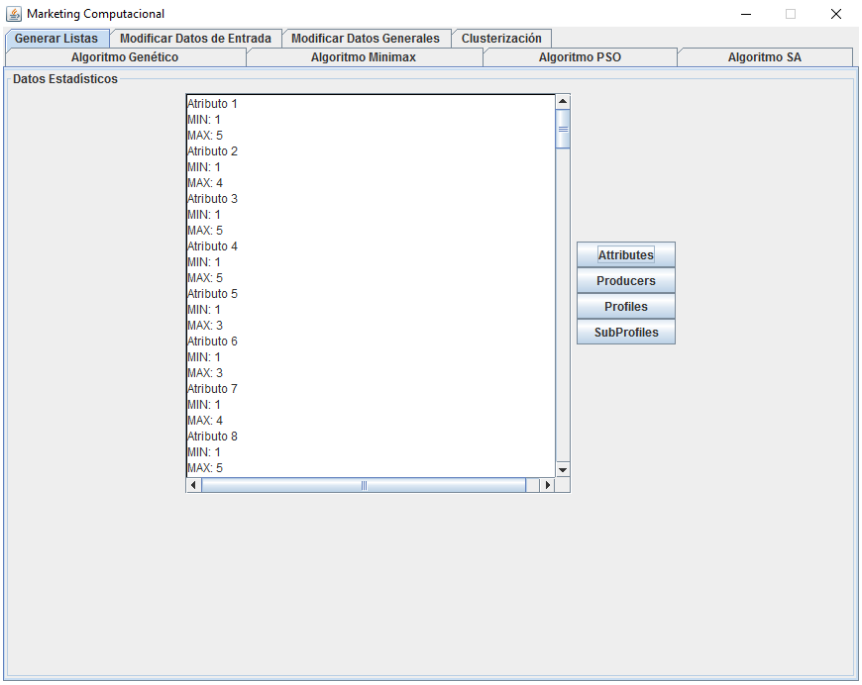


Figura 12 Ejemplo de lista de atributos generados

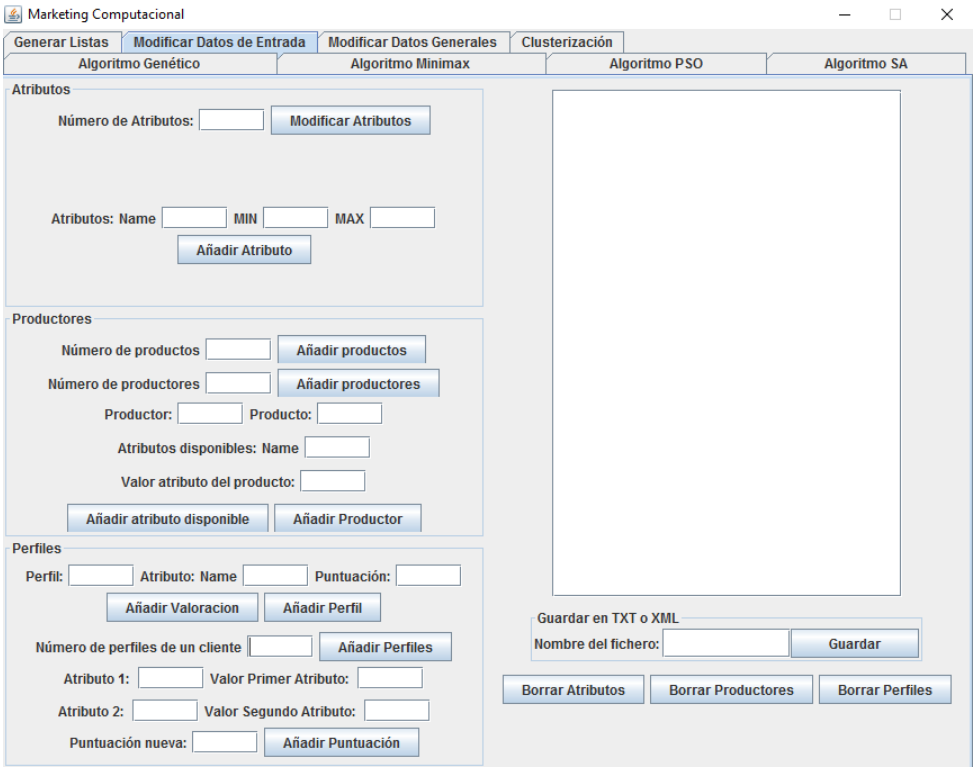


Figura 13 Modificación de datos de entrada

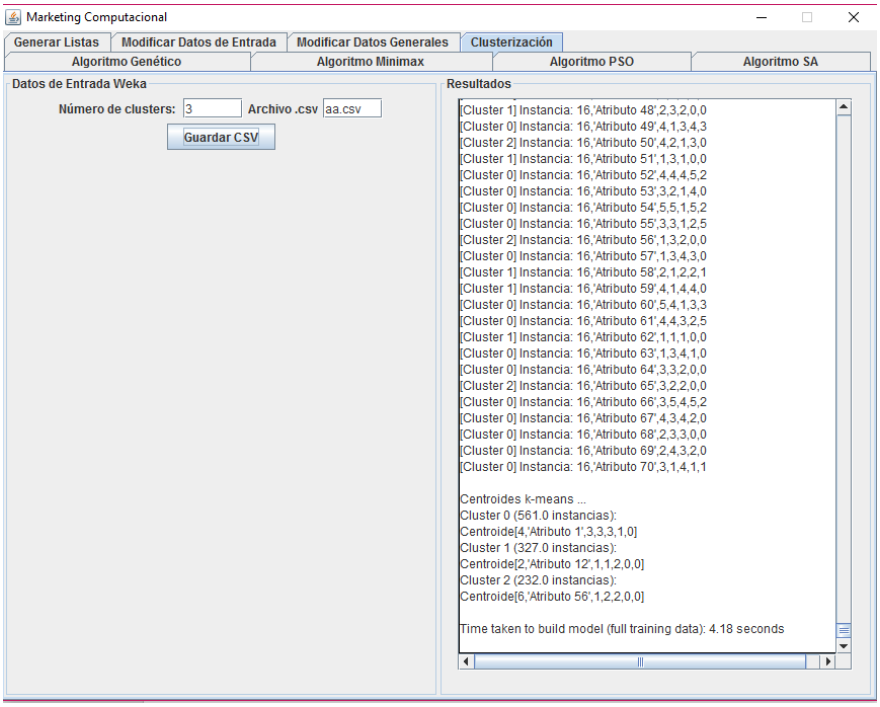


Figura 14 Ejemplo de clusterización

7.4.2 Interfaz para Android

La interfaz gráfica de la aplicación para Android es diferente a la de escritorio, ya que esta es una aplicación para móvil. Mostramos algunas imágenes de ejemplo que nos permiten ver cómo se realizan las ejecuciones, y cómo se introducen los datos de entrada en la aplicación.

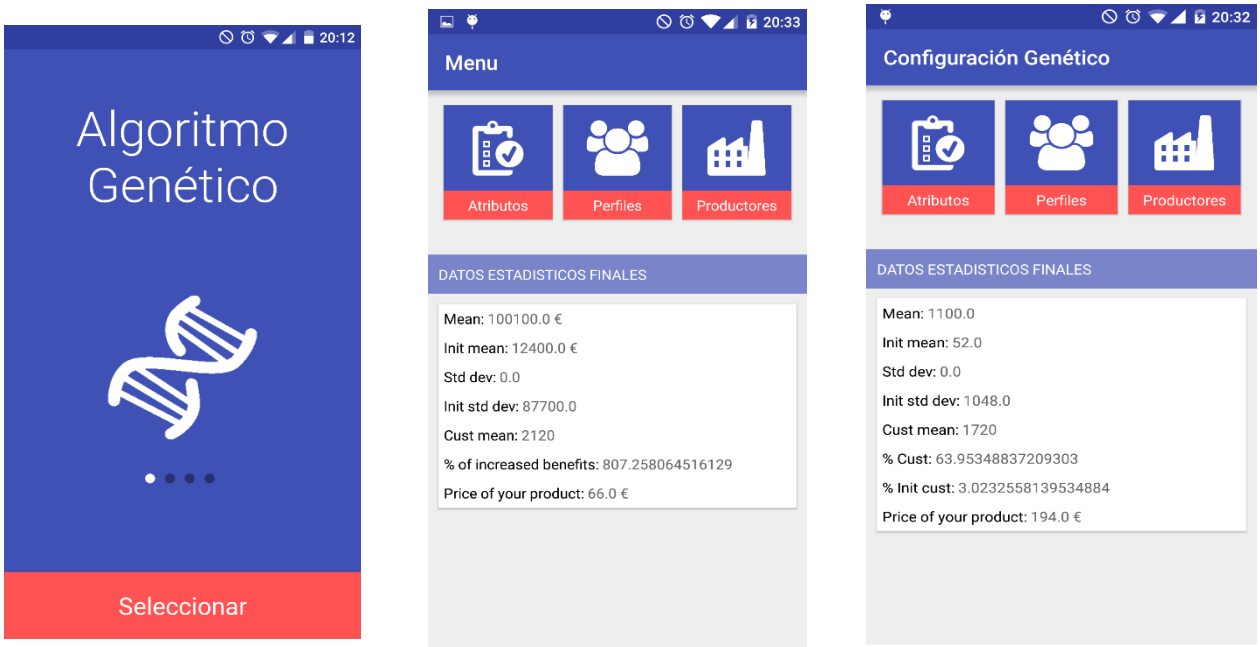


Figura 15 Ejemplo de ejecución Algoritmo Genético en Android

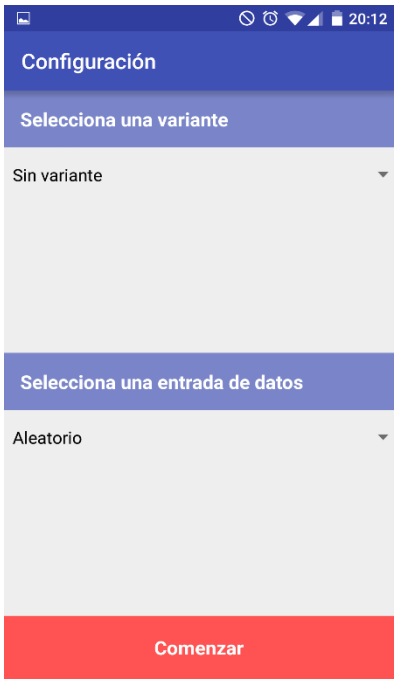


Figura 16 Ejemplo de configuración de variantes en Android

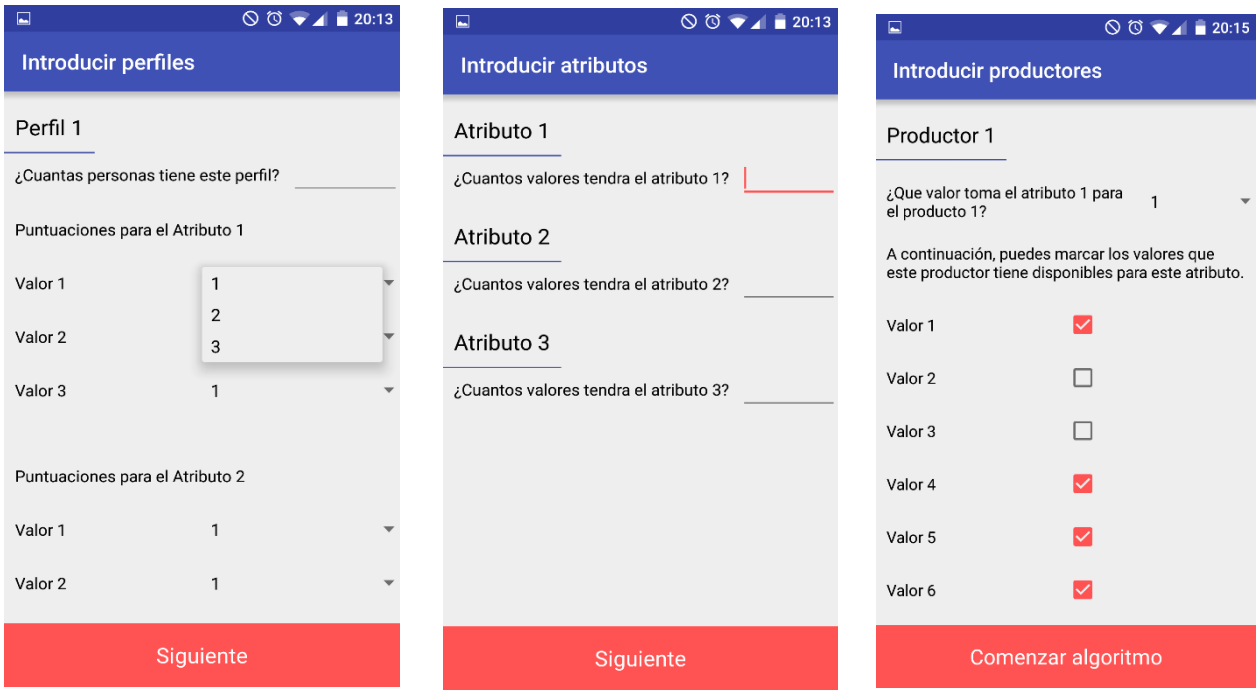


Figura 17 Modificación de datos de entrada en Android

8. Encuestas

En este apartado explicaremos cómo hemos llevado a cabo las encuestas que hemos realizado para obtener la instancia a resolver en nuestro caso de estudio, y cuáles son los resultados que hemos obtenido.

8.1 Plataforma

Para generar las encuestas de nuestro proyecto sobre un producto determinado, utilizamos la plataforma web de *Feebbo*.

Feebbo [1] es una empresa que se encarga de ofrecer una herramienta para que puedas crear tu propio estudio de mercado, a través de sus encuestas. Esta empresa bonifica a cada uno de sus clientes que participen en diversas encuestas de estudios de mercados. Además, también tienen la opción de rellenar encuestas de entretenimiento.

Lo más interesante de esta empresa, es que se puede ver la evolución sobre los estudios de mercado[7] a través de su plataforma, el número de personas que está realizando la encuesta, los últimos cambios en detalle, etc.

Para hacer tus estudios de mercado o poder acceder a estudios online que ya se habían realizado, tienen diferentes tarifas en las que puedes elegir cuántas personas deseas que realicen tu encuesta.

En nuestro caso, contactamos por correo con la empresa para saber si podrían ayudarnos de alguna forma con las encuestas que queríamos hacer, sin fines lucrativos. Satisfactoriamente, recibimos una contestación del jefe de *Feebbo* (Goyo Hernández), que estaba muy interesado en colaborar con nuestro trabajo de fin de grado. De hecho, su empresa suele apoyar este tipo de proyectos en diferentes universidades a nivel internacional.

Esencialmente, el acuerdo que tuvimos con *Feebbo* fue que nos facilitaban su plataforma (y también a sus clientes) para realizar las encuestas que nos hagan falta para el proyecto, a cambio de que podamos darle una solución a un problema dado que tenían, en este caso, poder clasificar a sus encuestados en clusters (explicado más adelante). En conclusión, el trato fue de información por información.

[7] Estudios de mercado de Feebbo: http://www.feebbo.com/feebbo/estudiosmercado

8.2 Organización de las Encuestas

Para la elaboración de las preguntas que utilizaríamos en nuestro caso de estudio, elegimos como tipo de producto a tratar de *vender* los teléfonos móviles. Posteriormente realizamos una investigación de las características, comunes o no, que tienen en general los móviles, y cuáles o cuántas de estas características son las que más le importan a la gente a la hora comprar uno. Investigamos sobre telefonía móvil en páginas como Apple [26], Samsung [27], MediaMarkt [28], Fnac [29], PC Componentes [30], etc.

Hicimos 63 preguntas sobre móviles a las cuales tuvimos que añadir preguntas sobre personalidad con el fin de que Feebbo pueda mostrar una respuesta a los encuestados sobre su tipo de personalidad basada en las características de móviles de su preferencia.

Además, dividimos la encuesta en otras tres para que no sea tediosa y resulte más dinámica para el encuestado (Feebbo nos indicó que, en encuestas demasiado largas, los encuestados terminan respondiendo al azar, lo que quita cualquier validez a los resultados).

Las preguntas que hicimos fueron de este estilo:

1. ¿Comprarías un Smartphone de una marca poco o nada conocida?
    a. Seguro
    b. Muy probable
    c. Poco probable
    d. Nunca
    e. Indiferente

2. ¿Cuál es tu valoración sobre Apple? De 0 (Menor valoración) a 5 (Mayor valoración)
    a. 5
    b. 4
    c. 3
    d. 2
    e. 1
    f. 0

3. ¿Cuál es tu valoración sobre Samsung? De 0 (Menor valoración) a 5 (Mayor valoración)
    a. 5
    b. 4

c. 3
d. 2
e. 1
f. 0

4. ¿Cuál es tu valoración sobre Huawei? De 0 (Menor valoración) a 5 (Mayor valoración)
    a. 5
    b. 4
    c. 3
    d. 2
    e. 1
    f. 0

5. ¿Cuál es tu valoración sobre BQ? De 0 (Menor valoración) a 5 (Mayor valoración)
    a. 5
    b. 4
    c. 3
    d. 2
    e. 1
    f. 0

6. ¿Cuál es tu valoración sobre las nuevas marcas chinas? De 0 (Menor valoración) a 5 (Mayor valoración)
    a. 5
    b. 4
    c. 3
    d. 2
    e. 1
    f. 0

7. Para mi Smartphone prefiero un diseño…
    a. Elegante
    b. Juvenil
    c. Moderno
    d. Indiferente

8. ¿Qué dimensiones prefieres que tenga la pantalla de un Smartphone?
    a. Grande (más de 5’’)
    b. Mediano (4’’ - 5’’)
    c. Pequeña (menos de 4’’)

d. Indiferente

9. ¿Qué sistema operativo te gusta más?
    a. iOS
    b. Android
    c. Windows phone
    d. Symbian
    e. Indiferente

10. ¿Cómo de importante consideras que tenga una gran gama de accesorios visuales? (Ej. Fundas, carcasas...)
    a. Muy importante
    b. Bastante importante
    c. Importante
    d. Algo importante
    e. Poco importante
    f. Indiferente

Las preguntas relacionadas con la personalidad que teníamos que añadir a nuestra encuesta, a petición de Feebbo aunque no aportasen información relevante para nuestro estudio, fueron como éstas:

1- ¿Cómo te defines a ti mismo?

a. Algo tímido, solidario y tranquilo. 2
b. Simpático divertido y entusiasta. 3
c. Distraído, pesimista y miedoso. 1

2- ¿Qué es lo que más te gusta de ti mismo?

a. Algunos rasgos físicos. 2
b. Todo. Estoy muy satisfecho con lo que tengo. 3
c. Quizás algo de mi personalidad, pero poco o casi nada. 1

Cada una de las posibles respuestas a cada pregunta tenía una puntuación diferente. La personalidad que se obtenía al final de la encuesta dependía de la puntuación que se obtuviera.

La lista completa de preguntas se adjunta en un documento por separado junto con la memoria.

8.3 Resultados de las Encuestas

Finalmente, Feebbo nos proporcionó los resultados de la encuesta (en este caso se particionaron en tres encuestas parciales) en archivos csv. El primer archivo, de sintaxis, contenía las preguntas de la encuesta y sus respuestas (cada respuesta tenía asignada un código). El segundo archivo, de datos, contenía los datos del encuestado y sus respectivas respuestas de la encuesta (especificadas con códigos).

resp_preg	Codigo	nombre resp
P1	531075	""Seguro""."
P1	531077	""Muy probable""."
P1	531079	""Poco probable""."
P1	531081	""Nunca""."
P1	531083	""Indiferente""."
P2	531085	""5""."
P2	531087	""4""."
P2	531089	""3""."
P2	531091	""2""."
P2	531093	""1""."
P2	531095	""0""."
P3	531097	""5""."
P3	531099	""4""."
P3	531101	""3""."
P3	531103	""2""."
P3	531105	""1""."
P3	531107	""0""."
P4	531109	""5""."
P4	531111	""4""."

Figura 18 Ejemplo de fichero Sintaxis

A	B	C	D	E	F	G
			531079	531087	531099	531111
IndentificadorUsuario	Edad	levelOfStudies	P1	P2	P3	P4
4775913-17991	21	primary_education	531075	531087	531099	531113
4766025-17991	27	university_masterdegree	531081	531093	531099	531113
4252803-17991	47	vocational_education	531079	531089	531099	531113
4775015-17991	19	secundary_education	531083	531085	531097	531111
4765651-17991	30	university_degree	531081	531095	531105	531119
4743829-17991	31	university_degree	531075	531089	531099	531113
4767119-17991	36	primary_education	531075	531085	531097	531109
4766047-17991	22	primary_education	531079	531089	531097	531115
4769263-17991	19	primary_education	531079	531087	531101	531115
4752609-17991	21	higher_education	531075	531089	531099	531111
4079514-17991	63	university_degree	531081	531087	531099	531117
4074992-17991	35	vocational_education	531079	531089	531101	531113
4769935-17991	23	higher_education	531081	531085	531097	531113
4760539-17991	23	vocational_education	531079	531087	531101	531113
4768801-17991	22	secundary_education	531081	531089	531097	531115
4273843-17991	40	university_masterdegree	531079	531085	531097	531111
4759953-17991	18	secundary_education	531075	531085	531097	531115
4676309-17991	28	secundary_education	531083	531089	531101	531115

Figura 19 Ejemplo de fichero Datos

Para tratar estos datos, tuvimos que adaptarlo al formato que ya teníamos predefinido en un documento de texto, teniendo en cuenta las respuestas que había

elegido cada persona encuestada, y así obtener los atributos, productores y perfiles de clientes.

Para crear los perfiles, decidimos tener en cuenta dos datos específicos de cada persona: la edad y el nivel de estudios.

Las edades de todas personas que realizaron la encuesta pertenecían a un rango de 14 a 74 años, de los cuales decidimos agruparlos en 3: primer rango de 14 a 29 años, segundo rango de 30 a 49 años y el tercer rango 50 a 74 años.

Los niveles de estudios de las personas que han realizado la encuesta son: doctorado, master, educación secundaria, educación primaria, grado en la universidad, educación vocacional, master en la universidad, educación superior.

Agrupamos los perfiles por su edad y, dentro de cada rango de edad, por cada nivel de estudio, por ejemplo:

Perfil 1: Edad 14 -29 años y doctorado.

Perfil 2: Edad 14 - 29 años y máster.

Perfil 3: Edad 14 – 29 años y educación primaria.

En cuanto a los valores de las valoraciones dadas por cada perfil, se asignan dependiendo del tipo de pregunta habiendo dos tipos posibles: preguntas sobre características específicas que no sean buenas ni malas en sí mismas (por ejemplo: iOs o Android), y preguntas que pregunten por algo intrínsecamente positivo, concretamente sobre cómo de importante considera el encuestado dicha característica ("poco importante", "algo importante", "importante", "bastante importante", "muy importante").

```
perfil 2 atributo23 4 3 1 2
perfil 2 atributo24 4 2 3 1
perfil 2 atributo25 4 2 2 3 1
perfil 2 atributo26 5 10 15 20 25 30 35
perfil 2 atributo27 4 8 12 16 20 24
perfil 2 atributo28 4 8 12 16 20 24
perfil 2 atributo29 3 2 1
perfil 2 atributo30 4 8 12 16 20 24 28
perfil 2 atributo31 5 10 15 20 25 30
perfil 2 atributo32 3 4 1 2
perfil 2 atributo33 2 4 3 1
perfil 2 atributo34 4 8 12 16 20 24
perfil 2 atributo35 2 4 3 1
perfil 2 atributo36 5 10 15 20 25 30 35
perfil 2 atributo37 2 4 6 8 10 12
perfil 2 atributo38 3 1 2
perfil 2 atributo39 3 1 2
perfil 2 atributo40 3 6 9 12 15 18 21
perfil 2 atributo41 5 10 15 20 25 30
perfil 2 atributo42 6 12 18 24 30 36 42
```

Figura 20 Ejemplo de perfiles en formato txt

El número de personas que realizaron las tres encuestas parciales fueron 535, 1500 y 1502 respectivamente.

En el siguiente apartado se describirán los experimentos en los que los datos de entrada de nuestros problemas fueron los generados a partir de los resultados de dicha encuesta.

8.4 Análisis de la Competencia

Después de haber realizado una investigación sobre telefonía móvil, a continuación listamos todos aquellos móviles que hemos utilizado para modelizar la competencia del producto que hemos diseñado con la aplicación. También mencionaremos sus características, para poder observar las diferencias que hay entre cada uno de ellos. Toda esta información la hemos recopilado de la página Ubergizmo [31].

Las características que utilizaremos de cada móvil son las que mismas sobre las que se han preguntado en las encuestas. Los móviles que hemos seleccionado son: iPhone 6s, Samsung Galaxy s6, BQ Aquaris M5.5, One Plus 2, Huawei P8 Lite, Sony Xperia Z4, Nexus 5X, Meizu M2 Note, LG G4 y Nokia Lumia 730.

Ejemplo de teléfono móvil considerado, junto con algunas de sus especificaciones técnicas concretas:

Iphone 6s:

- Dimensión de pantalla: 4.7'' mediana (4''- 5'') sobre grande (más de 5'') o pequeña (menos de 4'').
- Sistema operativo iOS sobre Android, Windows Phone o Symbian.
- Batería no extraíble sobre si extraíble.
- Forma de pantalla táctil sobre teclado deslizante, pantalla y teclado.
- Tipo de SIM: Nano SIM sobre SIM normal, Micro SIM.
- Botones de navegación táctiles sobre físicos.
- Tacto de tapa trasera liso sobre suave o rugoso.
- Material del móvil: metal sobre cristal o plástico.

Los demás móviles con sus respectivas características se adjuntan en un documento separado junto con la memoria.

9 Resultados de la Aplicación

En este apartado se describe los resultados finales de los experimentos realizados tanto para el caso de estudio real sobre móviles como para instancias aleatorias, y las explicaciones correspondientes sobre dichos resultados.

Después de implementar todos los algoritmos mencionados y generar un documento de texto con los resultados de las encuestas proporcionadas por Feebbo, el cual proporciona los datos de las instancias de nuestros problemas que queremos resolver en relación a nuestro caso de estudios sobre móviles, mostramos los gráficos que reflejan las ejecuciones de cada uno de los algoritmos y sus respectivos resultados.

Las instancias aleatorias de ambos problemas a resolver en nuestros experimentos se construyeron de la siguiente forma. Se consideran 63 atributos con 5 valores posibles para la construcción de cada una de las instancias PDO, mientras que 10 atributos con 2 valores posibles se utilizaron para SPDG. En ambos casos, los perfiles de cliente se crearon de esta manera. La valoración que cada perfil de cliente da a cada valor del atributo es siempre entre 0 y 5. Inicialmente, cuatro perfiles fueron creados al azar. A continuación, para cada uno de ellos creamos dos variantes adicionales.

Cada variante toma inicialmente las mismas valoraciones que el perfil del que proviene, y a los siguientes 33% de las valoraciones se dan nuevos valores aleatorios. Finalmente, se añadieron cuatro perfiles de clientes adicionales completamente al azar. Estos 16 perfiles de clientes intentan simular la forma de preferencias en un mercado viable: la mayoría de los perfiles de los clientes se reúnen en pocos grupos de preferencia similar, y hay algunos de otros tipos de clientes aislados.

Además, 10 productores fueron considerados en casos de PDO. Para el 90% de los atributos, todos sus valores de atributo estaban disponibles para ser seleccionados por cualquier productor. Sin embargo, para el resto de atributos, sólo un valor de atributo estaba disponible para todos los productores, mientras que el 33% de los productores puede tener más de un valor de atributo disponible (con una probabilidad del 50% para cada valor posible del atributo). En cuanto al problema SPDG, sólo se utilizaron 2 productores, y todos los valores de todos los atributos disponibles para ambos.

En PDO, los productos producidos inicialmente por cada uno de los 9 productores oponentes se crean de acuerdo con el siguiente algoritmo voraz. Para cada productor, se seleccionan al azar cuatro perfiles de clientes. A continuación, para cada atributo y para cada uno de sus valores de atributos, añadimos las valoraciones de este valor de atributo en estos cuatro perfiles de clientes.

Identificamos el valor del atributo cuya adición era el más alto, y este valor de atributo fue el valor asignado al atributo correspondiente del productor.

A continuación, el algoritmo genético, PSO y SA se llevaron a cabo para encontrar un producto bueno para el productor 1. Para cada atributo, la probabilidad de mutación era 0.01, y se aplicó un cruce uniforme. Una población inicial de 20 soluciones posibles (es decir, productos para el productor 1) se creó utilizando el algoritmo voraz previo, y el algoritmo genético, PSO y SA fue ejecutado cien veces.

Para el PDO, el productor 1 tiene la ventaja de diseñar primero su producto. Con el fin de comparar de forma equitativa el rendimiento del algoritmo genético, PSO y SA con el rendimiento de la estrategia voraz para resolver el problema PDO, se guardan los resultados iniciales obtenidos por el voraz para hacer una comparativa cuando terminen los diferentes algoritmos.

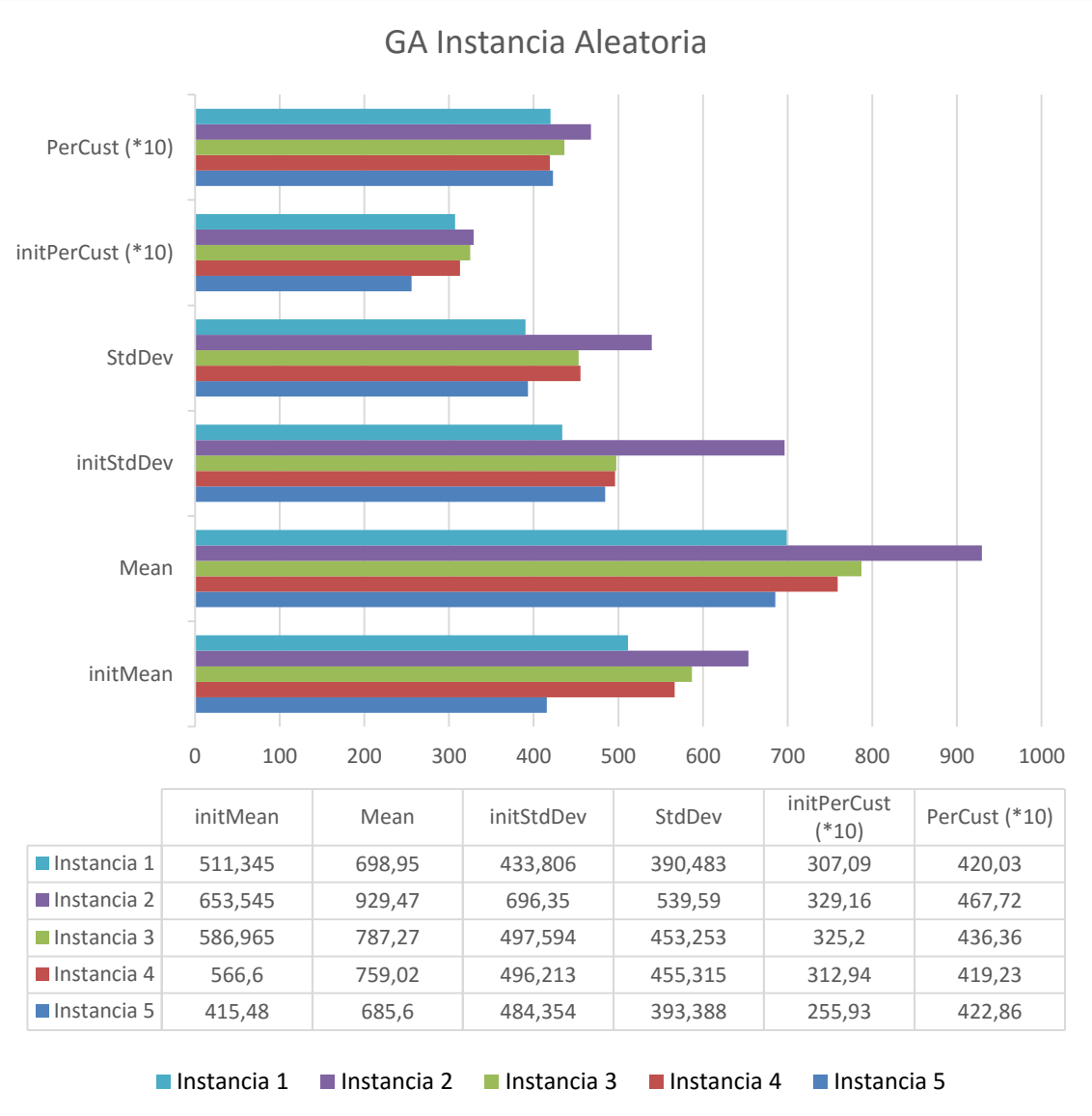
Para SPDG, los productos producidos inicialmente por los dos productores fueron creados al azar. Además, D = 1 (número de atributos modificables), tp = 5 (número del último turno jugado), tf = 5 (número de turnos totales del juego). La estrategia es que para cada turno del juego donde se mueve, se ejecuta un algoritmo minimax con poda alfa y beta con cierta profundidad $d1 \in \{4,6,8\}$. Para el productor 2, se utilizó el mismo método, aunque sus profundidades son $d2 \in \{1,2\}$. Para ambos productores, el valor heurístico tomadas en las hojas del árbol explorado por el algoritmo minimax fue el número de clientes recogidos durante todas las vueltas recorridas en la rama del árbol donde está la hoja.

Hemos generado la población inicial (sin utilizar clusterización) de los algoritmos GA y PSO de la siguiente manera. Se calcula un primer individuo que sea el resultado de un algoritmo voraz de entre todos los perfiles generados, de tal manera que el producto esté en la media de todos ellos. A partir de este producto generamos otros cercanos a él, y más adelante individuos aleatorios para añadirlos a la población. De esta forma partimos de una base con cierto criterio, que se basa en todos los perfiles a la vez.

Al usar clusterización para la generación de la población inicial, lo que hacemos es dividir el número de perfiles generados en un principio en X grupos. Y una vez tenemos definidos estos grupos, generamos para cada uno de ellos un producto que sea el resultado de un algoritmo voraz aplicado únicamente a ese grupo. Así en la población final tendremos tantos productos como grupos nos hayamos hecho a raíz de la clusterización, y todos ellos serán prometedores para algún sector de la población. Además, a esta población podremos añadir también el producto originado del algoritmo voraz de todos los perfiles para tener una población todavía más prometedora. Al tener tantos individuos diferentes, conseguidos a través de distintos sectores de los usuarios, tendremos más oportunidad de sacar un producto óptimo.

En todos los gráficos que se muestran a continuación el porcentaje inicial y final (initPerCust y PerCust respectivamente) se representan de esta forma: se multiplica el valor del porcentaje real por diez, con la finalidad de que se adapte al intervalo que hemos elegido para los gráficos y pueda visualizarse mejor.

- Para ejecuciones con datos generados aleatoriamente, se han realizado las siguientes pruebas:

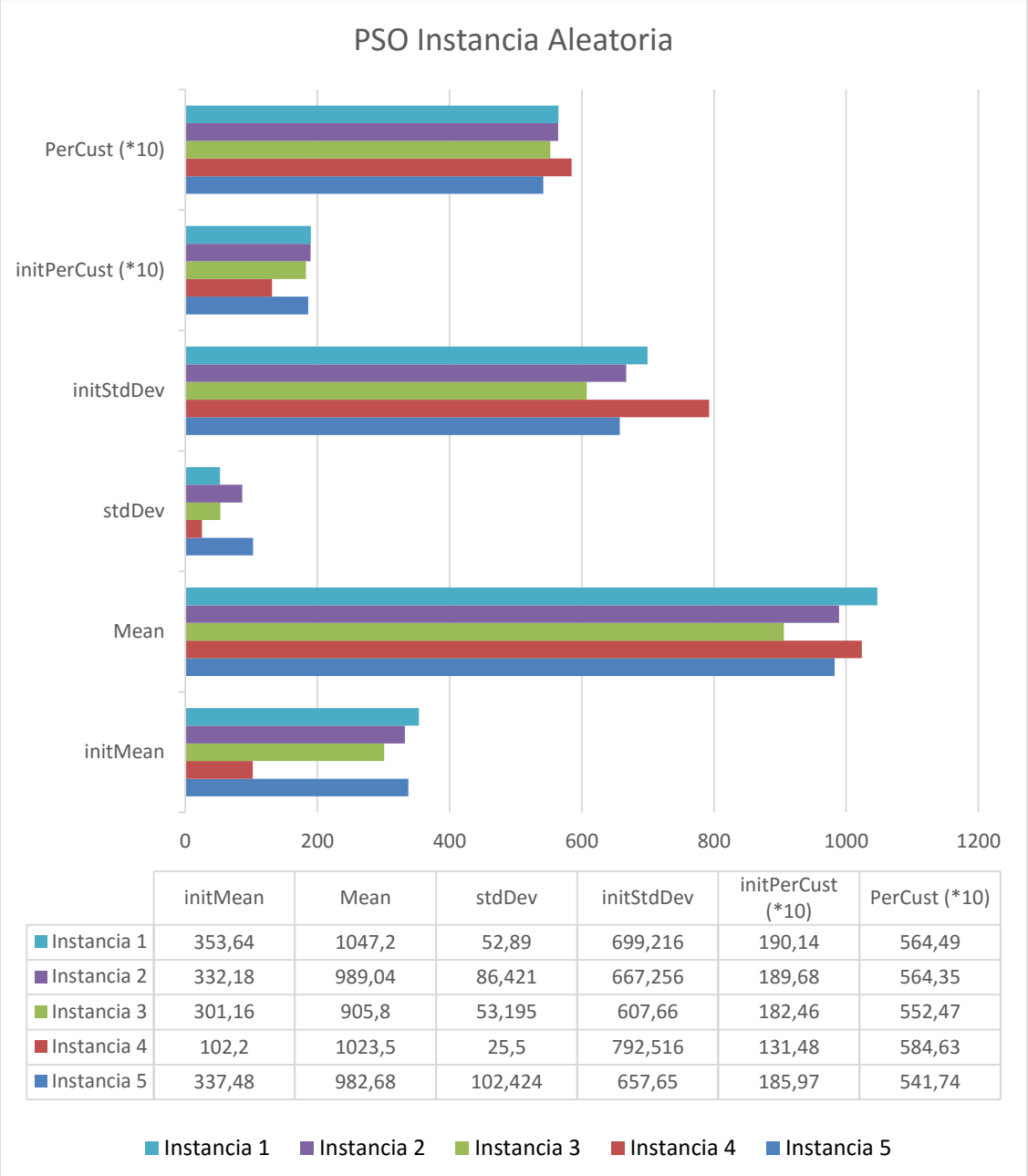


	initMean	Mean	initStdDev	StdDev	initPerCust (*10)	PerCust (*10)
Instancia 1	511,345	698,95	433,806	390,483	307,09	420,03
Instancia 2	653,545	929,47	696,35	539,59	329,16	467,72
Instancia 3	586,965	787,27	497,594	453,253	325,2	436,36
Instancia 4	566,6	759,02	496,213	455,315	312,94	419,23
Instancia 5	415,48	685,6	484,354	393,388	255,93	422,86

Figura 21 Ejecuciones Aleatorias Algoritmo Genético

En la Figura 21 se muestran los resultados del algoritmo genético en la resolución de cinco instancias aleatorias. Para cada una de ellas, se muestran los resultados de la media de diez ejecuciones sobre la misma instancia. Se muestran el porcentaje de clientes iniciales y finales obtenidos, los valores de media inicial y final de número de clientes obtenidos, desviación típica inicial y final de dicho valor.

Los resultados mejoran considerablemente entre un 10–12 de puntos porcentuales y en todas las ejecuciones se mantiene el porcentaje de mejora.

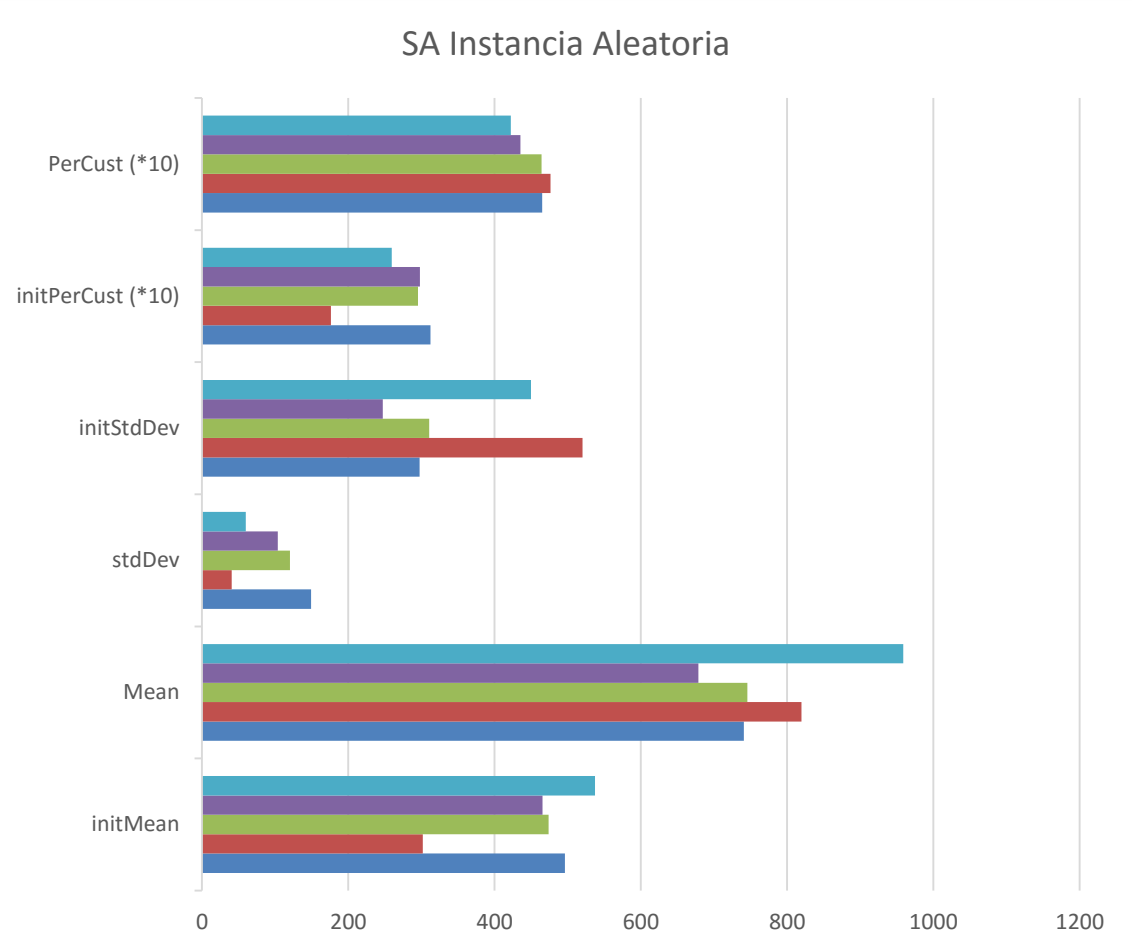


	initMean	Mean	stdDev	initStdDev	initPerCust (*10)	PerCust (*10)
Instancia 1	353,64	1047,2	52,89	699,216	190,14	564,49
Instancia 2	332,18	989,04	86,421	667,256	189,68	564,35
Instancia 3	301,16	905,8	53,195	607,66	182,46	552,47
Instancia 4	102,2	1023,5	25,5	792,516	131,48	584,63
Instancia 5	337,48	982,68	102,424	657,65	185,97	541,74

Figura 22 Ejecuciones Aleatorias Algoritmo por Enjambre de Partículas

En la Figura 22 se muestran los resultados de la ejecución de cinco instancias del problema mediante el algoritmo PSO. Se muestran los mismos datos considerados antes.

El algoritmo se muestra estable en sus resultados iniciales y finales para todas las instancias, consiguiendo siempre alrededor de 30 puntos porcentuales de mejora.

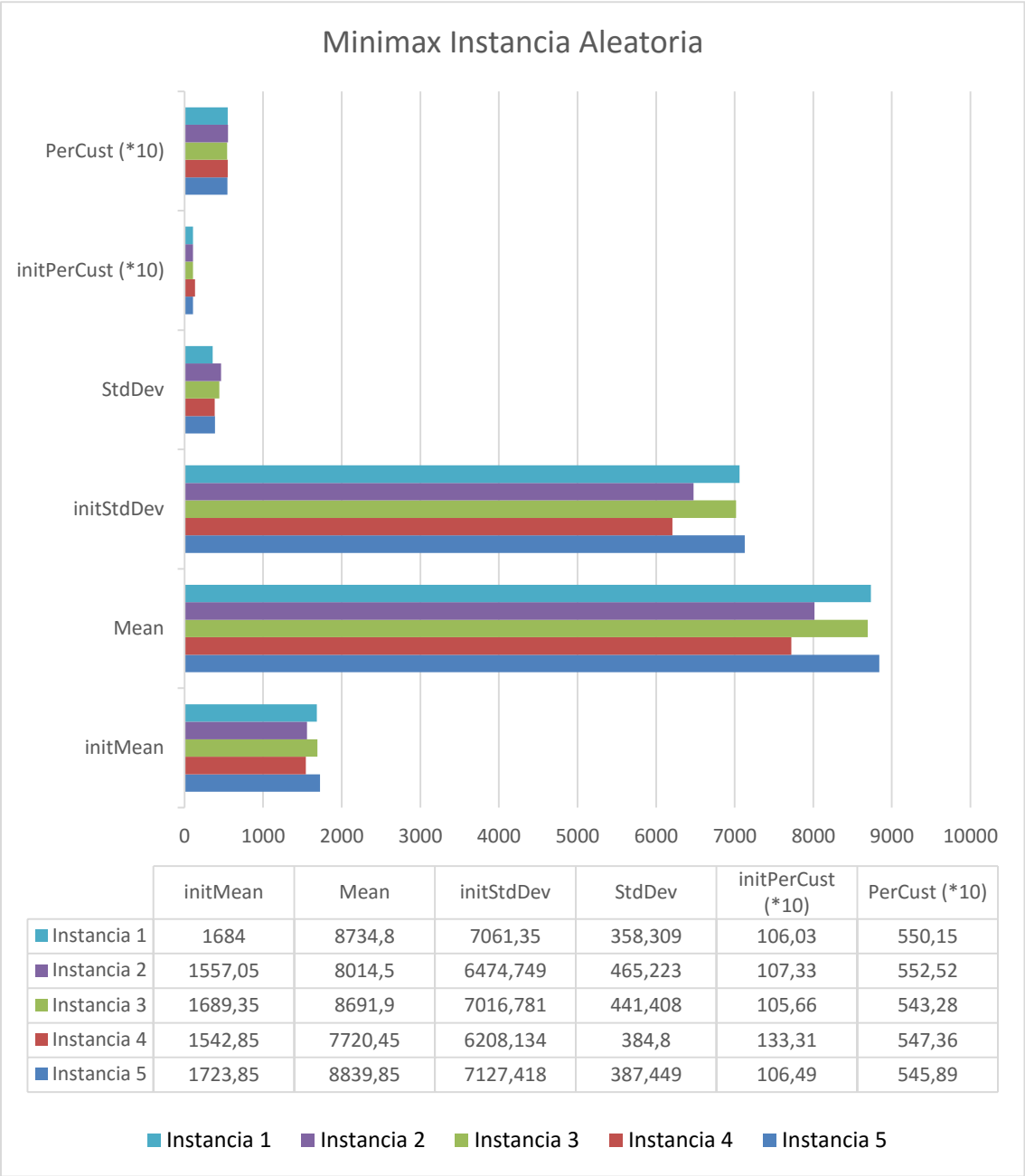


	initMean	Mean	stdDev	initStdDev	initPerCust (*10)	PerCust (*10)
Instancia 1	537,333	958,8	59,594	449,747	259,28	422,03
Instancia 2	465,467	678,5	103,43	247,131	297,89	435,15
Instancia 3	473,8	745,6	120	310,528	295,47	464,17
Instancia 4	301,8	819,8	40,662	520,282	176,2	476,63
Instancia 5	496,1	741	149	297,335	312,45	465,31

Figura 23 Ejecuciones Aleatorias Algoritmo de Enfriamiento Simulado

En la Figura 23 se muestran los resultados de la ejecución de cinco instancias del problema mediante el algoritmo de enfriamiento simulado.

Salvo en la instancia 4, todos los resultados son bastante estables y obtienen una mejora de entre 15 - 20 puntos porcentuales.



	initMean	Mean	initStdDev	StdDev	initPerCust (*10)	PerCust (*10)
Instancia 1	1684	8734,8	7061,35	358,309	106,03	550,15
Instancia 2	1557,05	8014,5	6474,749	465,223	107,33	552,52
Instancia 3	1689,35	8691,9	7016,781	441,408	105,66	543,28
Instancia 4	1542,85	7720,45	6208,134	384,8	133,31	547,36
Instancia 5	1723,85	8839,85	7127,418	387,449	106,49	545,89

Figura 24 Ejecuciones Aleatorias Algoritmo Minimax

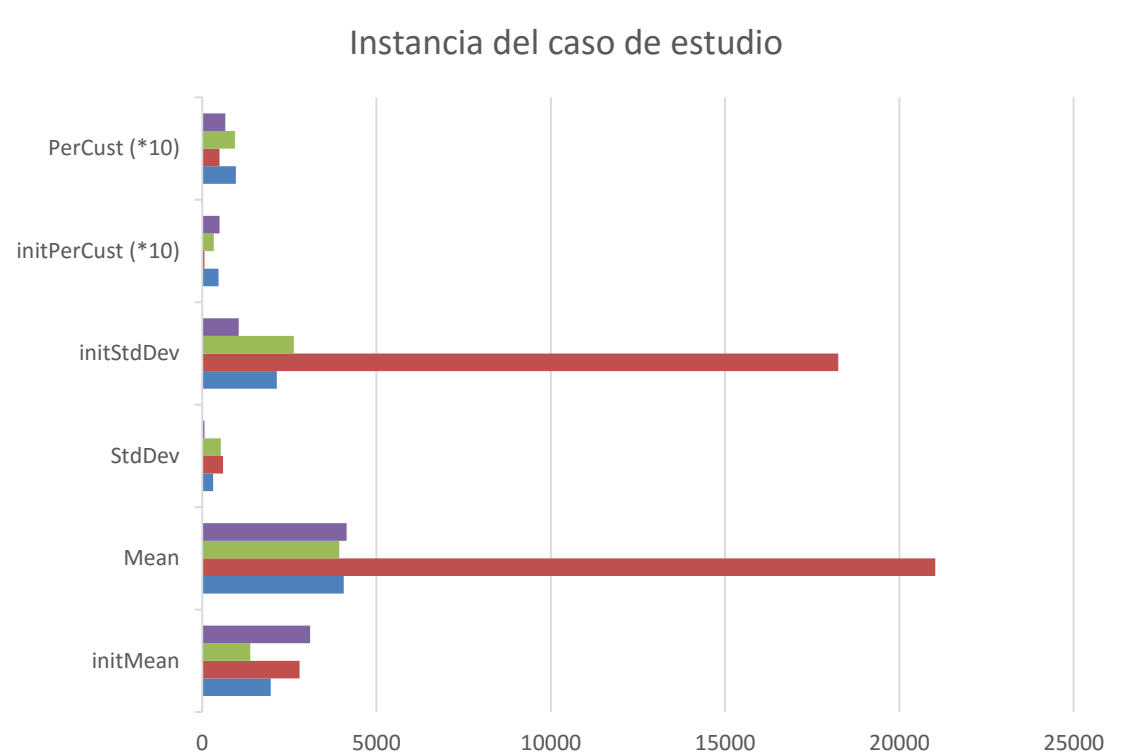
En la Figura 24 se muestran los resultados al ejecutar cinco instancias distintas del algoritmo minimax.

Para cada instancia hemos hecho diez ejecuciones y lo que mostramos en el gráfico es la media de esas diez ejecuciones. El algoritmo se muestra bastante estable para todas las instancias consiguiendo para cada una un margen de mejoras de entre 40 - 45 puntos porcentuales. Ya que para este algoritmo no creamos ninguna población inicial, no hace falta crear un primer individuo mediante un algoritmo voraz como en los otros casos.

Para nuestro algoritmo usamos la poda alfa-beta. La poda alfa-beta toma dicho nombre de la utilización de dos parámetros que describen los límites sobre los valores hacia atrás que aparecen a lo largo de cada camino. Esta búsqueda alfa-beta va actualizando el valor de los parámetros según se recorre el árbol. El método realizará la poda de las ramas restantes cuando el valor actual que se está examinando sea peor que el valor actual de α o β para MAX o MIN, respectivamente.

- Para ejecuciones realizadas sobre la instancia generada a partir de los resultados de las encuestas (es decir, nuestro caso de estudio sobre móviles), se muestran los siguientes resultados:

En cada uno de los algoritmos hemos realizado cinco ejecuciones, y lo que se muestra en este gráfico es la media de todas ellas.



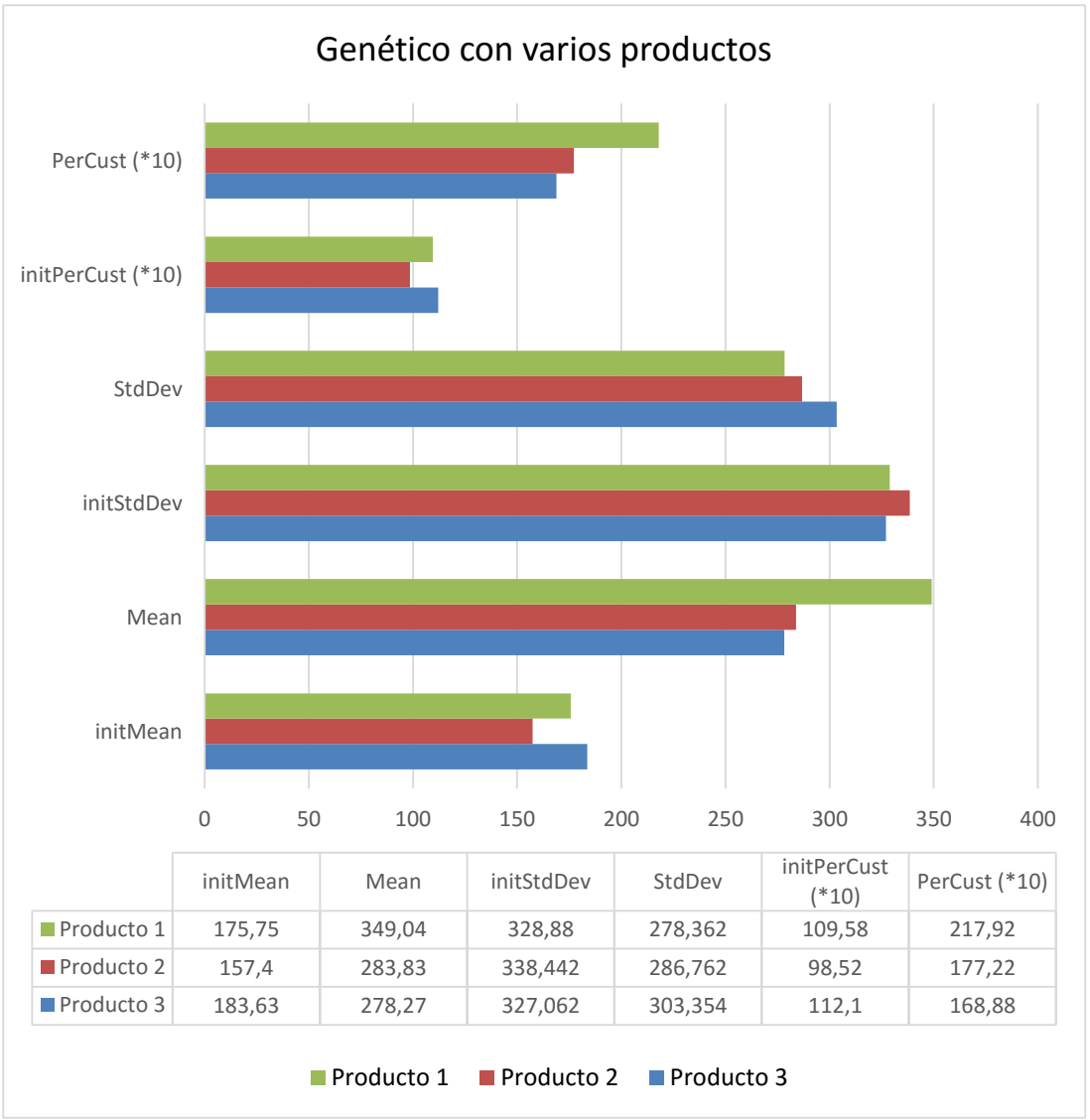
	initMean	Mean	StdDev	initStdDev	initPerCust (*10)	PerCust (*10)
SA	3098,4	4138,8	67,922	1044,228	495,38	662,12
PSO	1377,76	3929,2	533,62	2628,288	328,06	935,52
Minimax	2795,1	21026,9	598,01	18244,742	66,56	500,64
Genético	1967,87	4063,2	310,008	2141,332	468,54	967,44

Figura 25 Ejecuciones Datos Feebbo

En la Figura 25 se muestran los gráficos de los resultados obtenido al resolver nuestro caso de estudio real para cada algoritmo.

Se puede observar que en el caso del minimax tanto el Mean como la initStdDev están disparados. Esto se debe a que el valor que tomamos como referencia para calcular el Mean es la suma de los clientes obtenidos a lo largo de todos los turnos del juego, y no solo los clientes obtenidos en el último turno. Esto es así para poder obtener el producto que tenga mayor número de clientes a lo largo de un tiempo determinado. Esto crea una diferencia muy grande entre el initMean y el Mean, que es lo que a su vez ocasiona que se dispare el initStdDev.

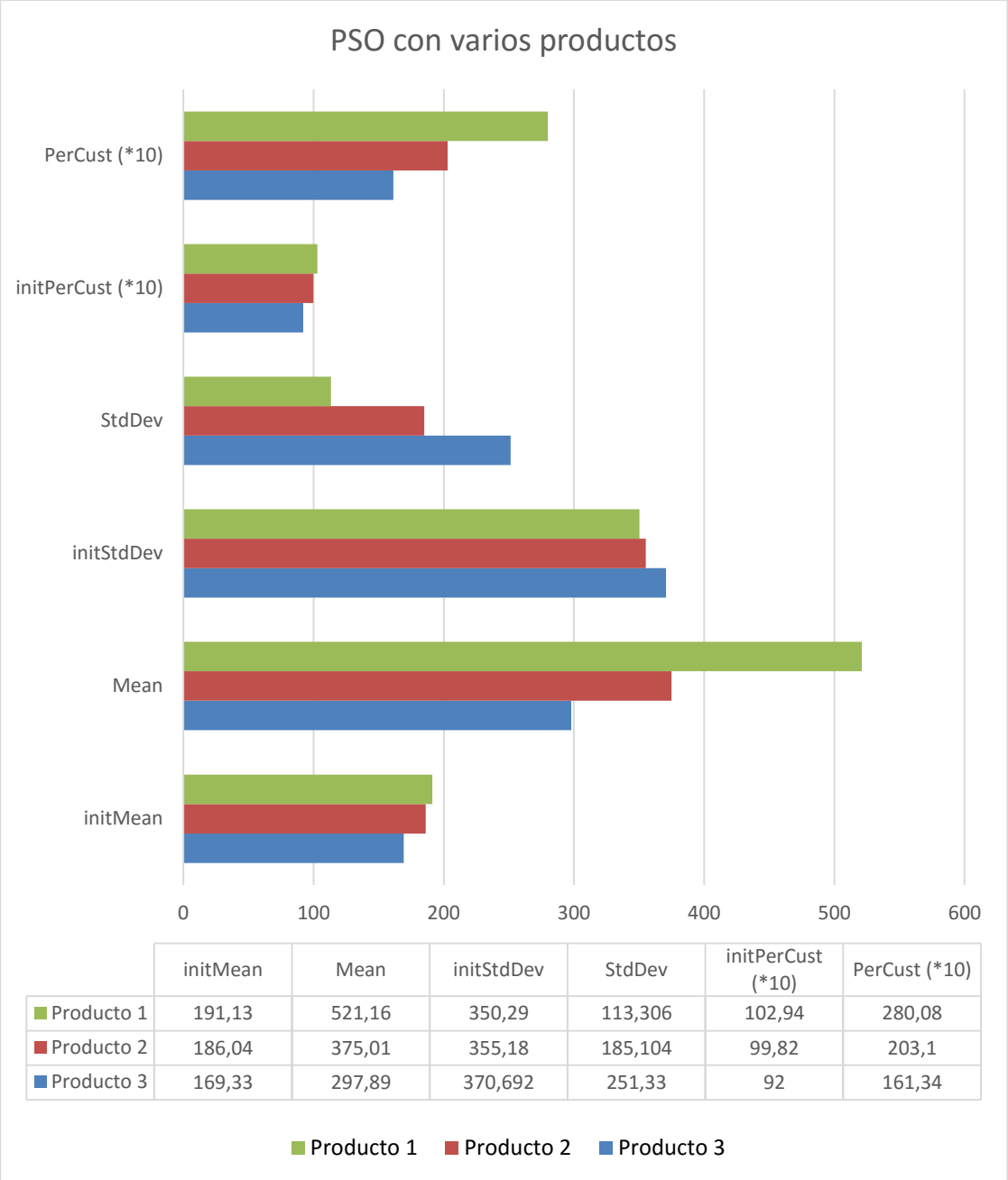
- Gráficos de ejecuciones con dos variantes: que cada productor tenga más de un producto.



	initMean	Mean	initStdDev	StdDev	initPerCust (*10)	PerCust (*10)
Producto 1	175,75	349,04	328,88	278,362	109,58	217,92
Producto 2	157,4	283,83	338,442	286,762	98,52	177,22
Producto 3	183,63	278,27	327,062	303,354	112,1	168,88

Figura 26 Ejecución del Algoritmo Genético con varios productos

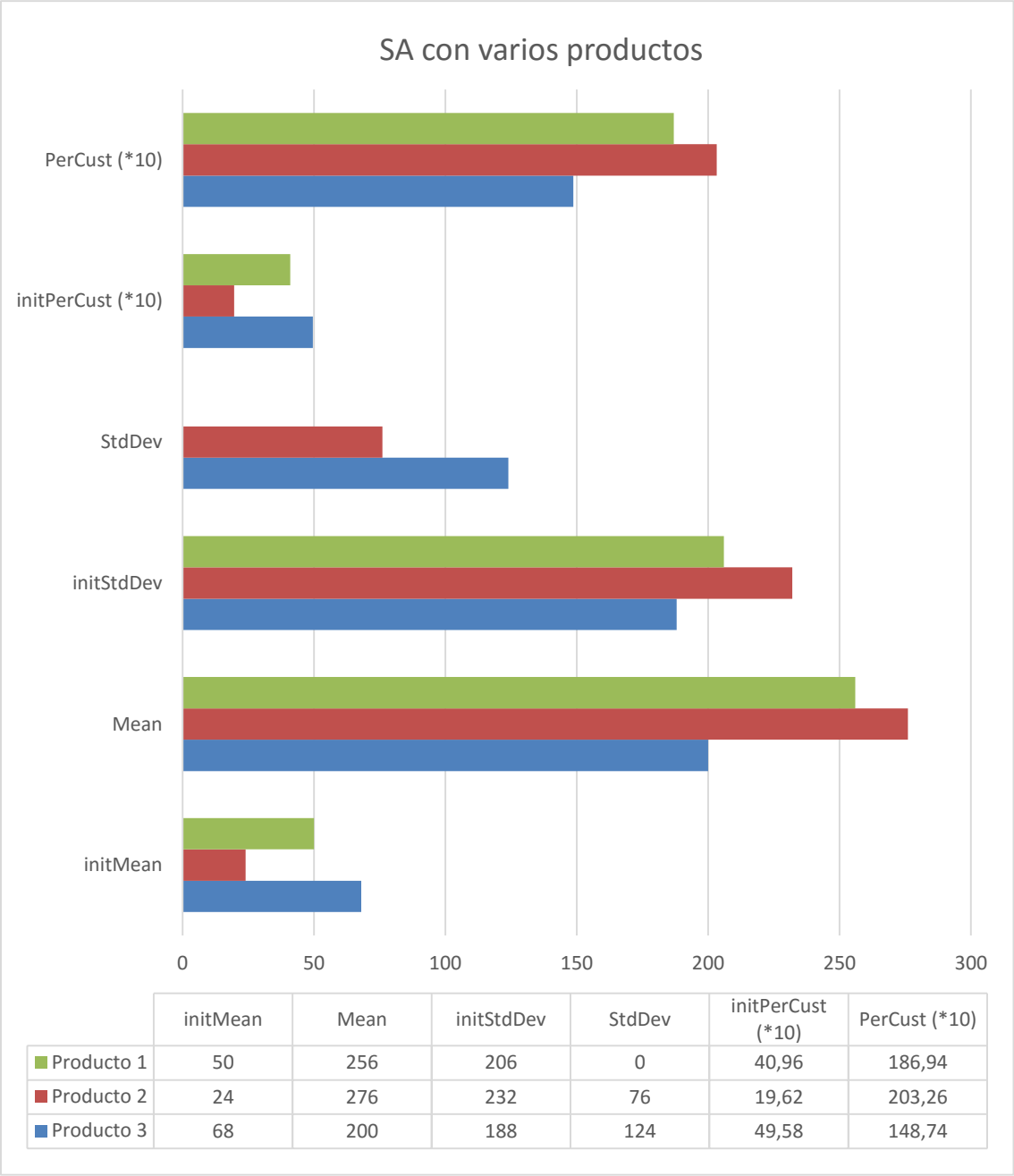
En la Figura 26 se muestran los resultados del algoritmo genético con la variante de PDO en la que cada productor puede vender más de un producto a la vez, en este caso hemos puesto como ejemplo tres instancias. Los gráficos muestran el total de clientes obtenidos entre los tres productos por cada una de las tres instancias.



	initMean	Mean	initStdDev	StdDev	initPerCust (*10)	PerCust (*10)
Producto 1	191,13	521,16	350,29	113,306	102,94	280,08
Producto 2	186,04	375,01	355,18	185,104	99,82	203,1
Producto 3	169,33	297,89	370,692	251,33	92	161,34

Figura 27 Ejecución del Algoritmo por Enjambre de Partículas con varios productos

En la Figura 27 se muestran los resultados del algoritmo PSO sobre dicha variante de PDO, bajo las mismas condiciones.

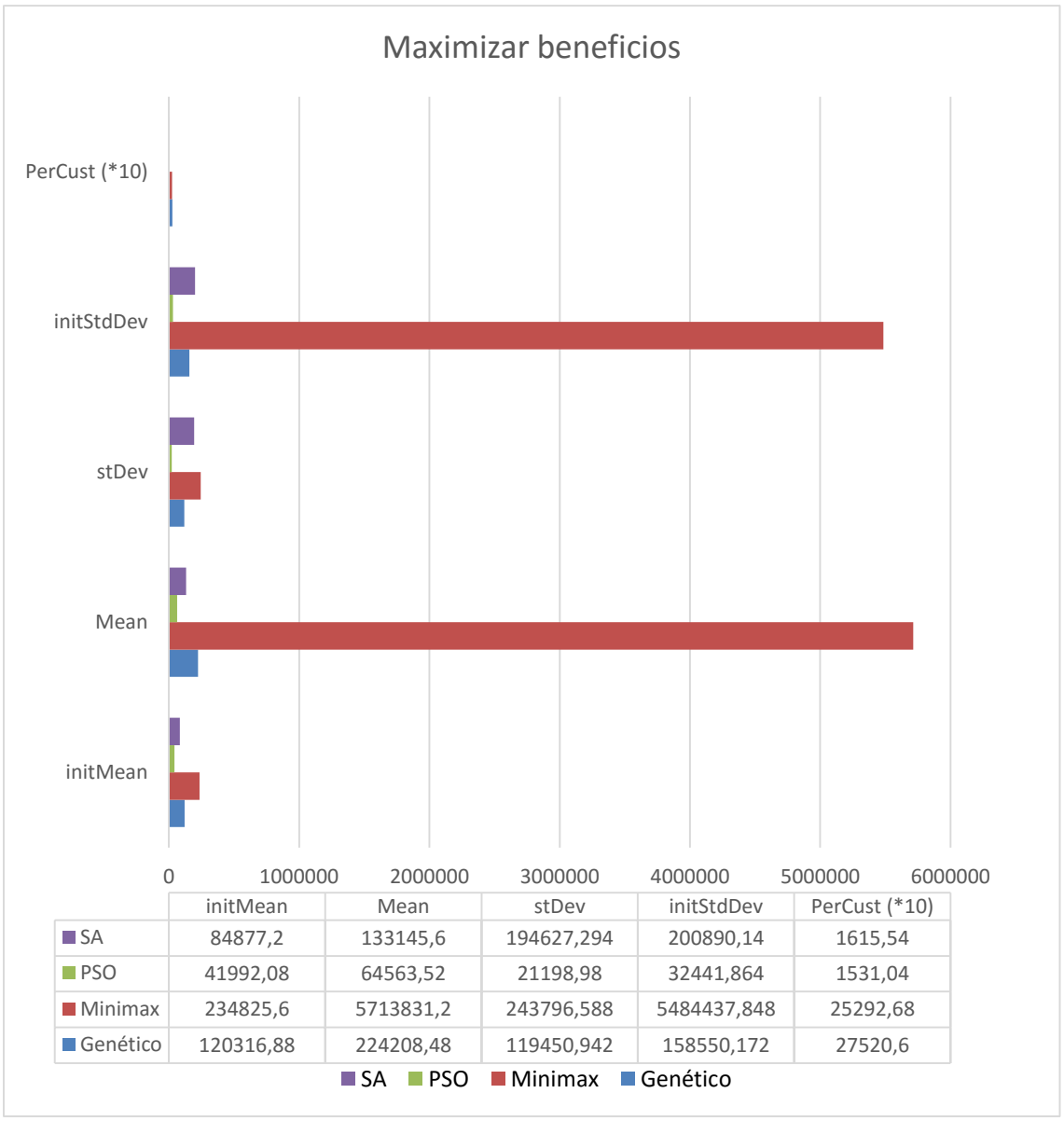


	initMean	Mean	initStdDev	StdDev	initPerCust (*10)	PerCust (*10)
Producto 1	50	256	206	0	40,96	186,94
Producto 2	24	276	232	76	19,62	203,26
Producto 3	68	200	188	124	49,58	148,74

Figura 28 Ejecución del Algoritmo de Enfriamiento Simulado con varios productos.

En la Figura 28 se muestran los resultados del algoritmo SA para la misma variante.

- Para ejecuciones en los que se maximice los ingresos:



	initMean	Mean	stDev	initStdDev	PerCust (*10)
SA	84877,2	133145,6	194627,294	200890,14	1615,54
PSO	41992,08	64563,52	21198,98	32441,864	1531,04
Minimax	234825,6	5713831,2	243796,588	5484437,848	25292,68
Genético	120316,88	224208,48	119450,942	158550,172	27520,6

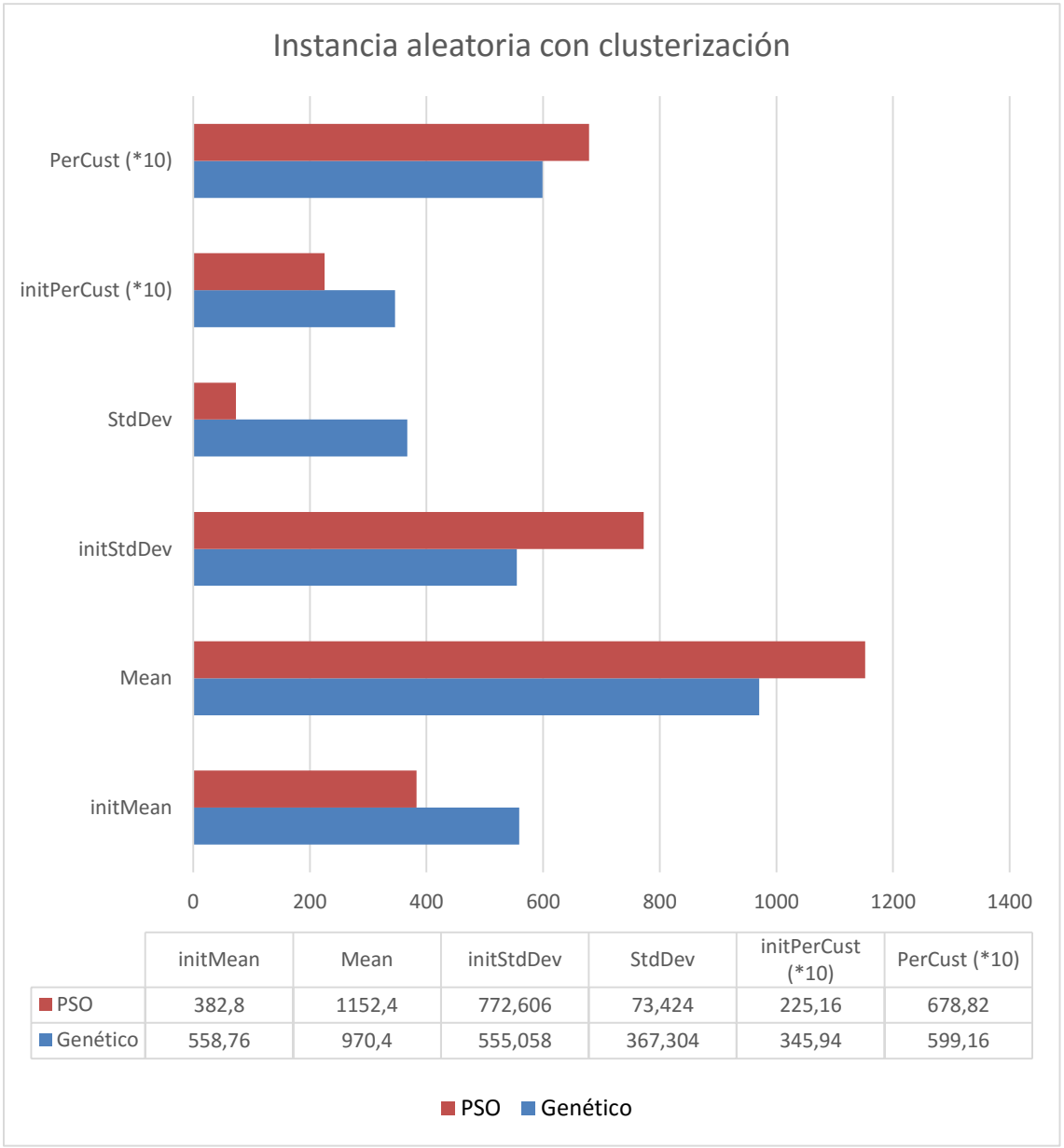
Figura 29 Ejecución de los cuatro algoritmos, en la variante de maximizar beneficios.

En la Figura 29 podemos observar cómo se comportan los cuatro algoritmos en la resolución de las correspondientes variantes de PDO y SPDG en las que el objetivo es maximizar beneficios en lugar de maximizar el número de clientes. En este gráfico, % de incremento indica el tanto por ciento de incremento de beneficios que ha conseguido el productor con respecto a su situación inicial.

En este gráfico volvemos a encontrar muy altos lo valores de Mean e initStdDev en comparación con los demás algoritmos, y la causa vuelve a ser la misma que en el

gráfico de la instancia del caso de estudio. Lo que ocurre es que estamos contabilizando la suma de los beneficios obtenidos en cada uno de los turnos del Minimax. Así el resultado de beneficios finales obtenidos es tan elevado, y por lo tanto el initStdDev también.

- Para ejecuciones con datos generados aleatoriamente y utilizando clusterización:



	initMean	Mean	initStdDev	StdDev	initPerCust (*10)	PerCust (*10)
PSO	382,8	1152,4	772,606	73,424	225,16	678,82
Genético	558,76	970,4	555,058	367,304	345,94	599,16

Figura 30 Ejecución de dos algoritmos utilizando clusterización

En la Figura 30 podemos observar cómo mejoran los resultados en dos algoritmos de ejemplo, en este caso PSO y Genético, utilizando el paso previo de clusterización para generar la población inicial del GA y el PSO.

Para este gráfico se ha usado la instancia número 1 usada para sacar los gráficos anteriores de instancias aleatorias. Se puede observar que los resultados son mejores que los obtenidos en los gráficos anteriores, los cuales no usan clusterización. El GA pasa de 42 puntos porcentuales sin clusterización a 60. En el caso de PSO pasamos de 56 puntos a 67,8.

- Ejecución del algoritmo minimax utilizando diferentes profundidades:

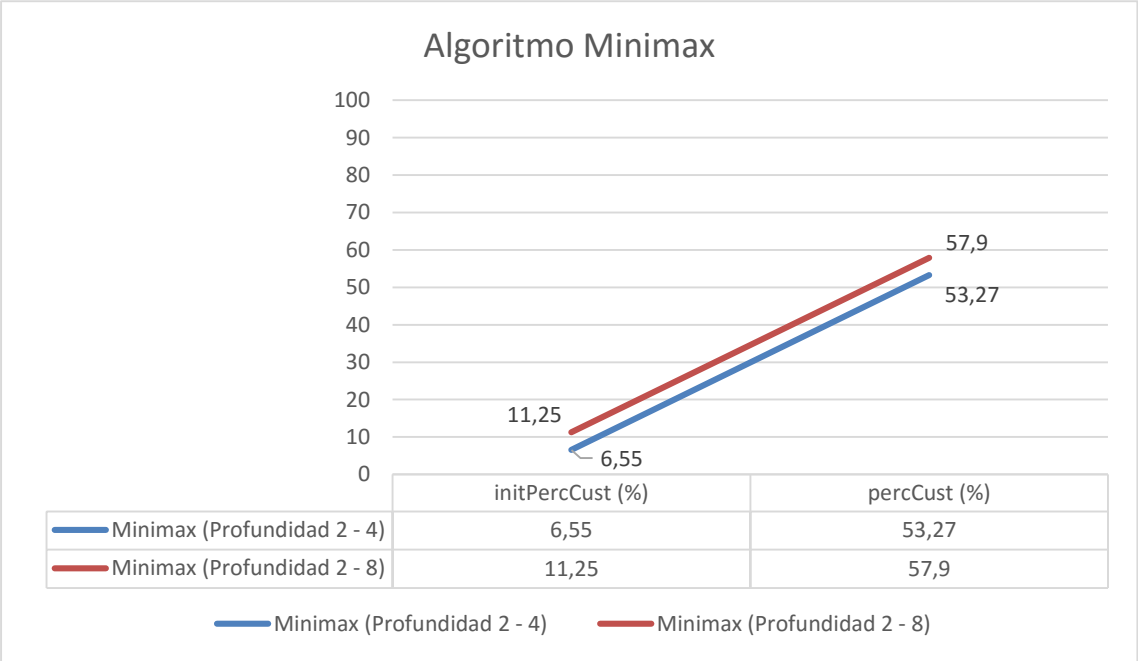


Figura 31 Ejecución del algoritmo minimax con distintas profundidades

En la Figura 31 podemos observar la ejecución del algoritmo Minimax con distintas profundidades para comprobar que, dependiendo de la profundidad que se use para cada competidor, se obtendrán mejores resultados o no.

10 Conclusiones

En este proyecto se han implementado algoritmos para resolver los problemas PDO, SPDG y lSPDG, tres problemas que tratan de encapsular objetivos esenciales en marketing: la selección de un producto que logre un número alto de clientes de forma inmediata (en PDO), y la selección de una estrategia de diseño de producto sostenible que logre un número determinado de clientes en un largo o corto plazo (en SPDG y lSPDG respectivamente). Para resolver estos problemas, hemos utilizado métodos heurísticos subóptimos, en lugar de utilizar algoritmos óptimos que resultarían intratables.

Desarrollamos experimentos con datos aleatorios y con datos obtenidos de un caso de estudio real sobre móviles.

Para llevar a cabo los objetivos que teníamos planteados hemos diseñado una interfaz multiplataforma para que los resultados se puedan visualizar claramente y se puedan introducir los datos a través de la aplicación. Además, en cuanto a la implementación de la aplicación, el código ha sido abstraído y bien organizado de tal manera que pueda ser reutilizable y se puedan añadir diferentes algoritmos con problemas nuevos.

Considerando los tres objetivos propuestos al principio del desarrollo del proyecto, hemos desarrollado un programa útil y eficiente que nos permite resolver nuestro objetivo principal: decidir las características del producto a vender para maximizar el número de clientes de forma inmediata o a largo plazo. Hemos llevado a cabo un caso de estudio real para aplicarlo a nuestra herramienta.

Para completar el tercer objetivo del proyecto, a partir de los resultados experimentales descritos en la sección anterior, exponemos nuestras conclusiones sobre los distintos algoritmos que hemos implementado.

Comenzaremos por el problema PDO y el *algoritmo genético*. Este algoritmo tiene unos resultados iniciales bastante estables. Aunque en algún caso sí que obtenemos algún resultado que se aleja de la media, es en las menores ocasiones, y parece depender mucho de la instancia con la que ejecutemos el algoritmo. Los resultados finales alcanzan casi siempre entre unos 40 y 45 puntos porcentuales, con lo que obtenemos resultados finales estables, aunque son unos de los más bajos de entre todos los algoritmos. Además el índice de mejora sobre los resultados de la población inicial del algoritmo es también el más bajo de todos los algoritmos, tan solo mejora una media de 12 puntos porcentuales por ejecución.

En el *algoritmo PSO* tenemos datos curiosos. Por ejemplo, el producto inicial de este algoritmo comienza teniendo un 17 % de los clientes iniciales, una de las más bajas situaciones iniciales, algo extraño ya que tanto el GA como el PSO obtienen su situación inicial de la misma manera y deberían tener puntuaciones iniciales parecidas. Esto nos hace pensar que podría haber un bug en el código el cual no ha sido encontrado, ya que los dos algoritmos generan la población de la misma manera, y uno comienza con un 17% del mercado, y otro con el 35%. Pensamos en la idea de un bug a pesar de que, tras estudiar la ejecución del algoritmo paso a paso con instancias pequeñas (lo suficiente para poder hacer el seguimiento manualmente), comprobamos que el algoritmo funciona correctamente.

Por último, vamos a analizar los resultados del *algoritmo de enfriamiento simulado*. Este algoritmo obtiene resultados normales, sin ningún tipo de anormalidad. Los resultados iniciales están en la media de los resultados obtenidos por todos los demás algoritmos y lo mismo ocurre con sus resultados finales. Como muestran los resultados, no es ni el que peor empieza, ni el que mejor acaba, pero siempre se queda entre los 45 puntos porcentuales, así que podemos decir que es un algoritmo estable.

Para el problema SPDG y el *algoritmo minimax*, el tanto por ciento de clientes conseguido en la situación inicial es bajo, en torno a 10 %, y sin embargo el *algoritmo minimax* obtiene una gran optimización del producto, consiguiendo alcanzar cerca de 55 % de los clientes del mercado. Hay que tener en cuenta que en este problema también se mejora el producto de la competencia, así que también es el más realista, dado que resulta difícil creer que un competidor no vaya a evolucionar su producto para obtener más beneficios que su oponente.

Este algoritmo es muy estable, habiendo una diferencia mínima de un 0.9 puntos porcentuales entre su mejor y su peor resultado.

Dado que el Minimax por un lado, y los algoritmos GA, PSO y SA por otro, resuelven problemas distintos es inútil comparar sus resultados. El Minimax trata de resolver un problema SPDG, mientras que los demás resuelven el problema PDO. Además estos algoritmos están utilizando instancias distintas, por lo que definitivamente no tiene sentido compararlos, ya que no aportaría ninguna información útil.

Ahora vamos a analizar los resultados de los mismos algoritmos, pero habiendo utilizado los datos de nuestro propio caso de estudio. Recordemos que esta instancia es construida a partir de los resultados de encuestas que solicitamos a la empresa Feebbo. Comentemos las grandes discrepancias con respecto a los casos de estudio de instancias aleatorias.

Por ejemplo, una de las cosas que más nos llaman la atención es que, excepto en el minimax, todos los demás algoritmos mejoran considerablemente (entre los 15 puntos porcentuales) sus situaciones iniciales y finales.

En el caso del Minimax es al revés, este empeora sus resultados finales en comparación con el caso de estudio con instancias aleatorias, pero lo curioso es que la mejora es la misma. En este caso, empieza 4 puntos porcentuales por debajo y termina igual, con el resultado final 4 puntos por debajo del caso de estudio aleatorio. Esto hace que la mejora sea la misma, e indica que el algoritmo es muy estable, y esta bajada en los resultados iniciales puede deberse simplemente al cambio de instancia utilizada, ya que la cantidad de datos que maneja es considerablemente diferente.

En los resultados del algoritmo SA ocurre lo mismo. Si bien hemos comentado que su resultado inicial y final aumenta, hay que recalcar que aumenta lo mismo que entre las instancias aleatorias. Y el índice de mejora entre el caso de estudio aleatorio y el caso de estudio real es el mismo, 17 puntos porcentuales. Esto indica lo mismo que en el minimax, que es un algoritmo estable y esta diferencia puede deberse al cambio considerable de la instancia utilizada.

Para el GA y PSO ocurre una anomalía. Ambos algoritmos dan resultados parecidos, cosa que no es de extrañar gracias a su gran parecido. Sin embargo, dichos resultados tienen algo inusual. Si bien los dos algoritmos comienzan con una situación inicial más alta que con el caso de estudio aleatorio, consiguen una situación final extrañamente alta, ambos algoritmos superan los 90 puntos porcentuales, en el caso del GA llega hasta los 96. No tenemos muy claro a qué se debe este aumento de la eficiencia del algoritmo, pero observando que se produce en los dos algoritmos que son similares podemos pensar que no se trata de un error en la ejecución del algoritmo, ya que tendría que haberse producido un error en ambos algoritmos que desemboquen en la misma repercusión, y eso es complicado. De los dos algoritmos el que mayor optimización realiza vuelve a ser el PSO. Esta anomalía podría deberse a la instancia utilizada, puede que los resultados obtenidos de las encuestas de Feebbo, o bien nuestra manera de convertir los datos de dichas encuestas en datos de entrada para el problema PDO no haya dado lugar a una instancia que capte fielmente la dificultad de lidiar con la competencia o de combinar apropiadamente los atributos de los productos, y que por ello, para la instancia obtenida, sea sorprendentemente fácil captar una cantidad enorme del mercado (cuesta creer que, si realmente existiera tal enorme nicho de demanda insatisfecho, las empresas que fabrican teléfonos móviles no se hubieran dado cuenta ya).

Después nos encontramos los resultados de los diferentes algoritmos usando la clusterización. El primero en analizar es el *GA*. En ambos casos (usando o no la clusterización para generar la población inicial) la mejora que nos encontramos es que, viendo los resultados iniciales, en el caso de la clusterización el tanto por ciento de

clientes obtenidos es un 10% superior, con lo cual el índice de clientes conseguidos al final del algoritmo es igualmente superior al algoritmo sin clusterización.

Con el algoritmo *PSO* nos encontramos el mismo comportamiento que en *GA*, el índice de mejora es igual en ambos casos, y cuando usamos la clusterización, conseguimos comenzar el algoritmo en una posición más ventajosa que sin él. Eso influye en que el resultado final también sea mejor.

Consideremos los resultados para las variantes que maximizan los beneficios en contraposición a los clientes. En el caso de PDO, podemos observar que el algoritmo SA y PSO obtienen un incremento de los ingresos parecido, en torno a 150 puntos porcentuales de mejora. Mientras que el GA llega hasta más de los 250 puntos.

En esta versión del problema el algoritmo que mejores resultados parece obtener es el Genético con una mejora de 275 puntos porcentuales. No sabemos muy bien por qué el PSO y el SA se distancian tanto en esta variante del algoritmo si en las ejecuciones para el PDO estándar no se llevan tanta diferencia.

Con respecto a los resultados obtenidos en la resolución de la variante en la que cada productor puede vender simultáneamente con varios productos, observamos las siguientes distinciones en los tres algoritmos donde se ha aplicado esta variante.

Si sumamos los puntos porcentuales obtenidos entre los tres productos, nos da una situación inicial acorde con los datos obtenidos en las ejecuciones estándar.

Con respecto a la situación final ocurre lo mismo: los puntos porcentuales obtenidos por los productos son inferiores a la ejecución estándar, pero si les sumamos los obtenidos por los tres productos vendidos conjuntamente, obtenemos incluso una mayor puntuación que en la versión normal. Esto se debe a que se aplica el algoritmo genético a tres productos distintos conjuntamente. Si somos capaces de optimizar un solo producto hasta cierto punto, y aplicamos esa misma técnica a otros dos productos más de manera conjunta y coordinada, obtendremos mejores resultados, obteniendo mejores productos, los cuáles se coordinan para repartirse el espacio de demanda.

En el último experimento, que corresponde a la ejecución del Minimax con distintas profundidades, se observa claramente que cuando ejecutamos el problema con una profundidad 2 para el oponente y 4 para nosotros, obtenemos una puntuación final de 53 puntos porcentuales. Sin embargo, si ejecutamos el algoritmo con una profundidad superior para nosotros (2 para el oponente, 8 para nosotros), obtenemos una puntuación final mejor, de 57 puntos porcentuales. Así comprobamos que, si vamos reduciendo la profundidad nuestra y ampliando la del oponente, los resultados serán peores para nosotros y mejores para el oponente. Por tanto, el resultado obtenido por el Minimax depende de la profundidad a la que estemos dispuestos a

llegar, sin olvidar que el tiempo de ejecución crece exponencialmente con la profundidad.

Finalmente, llegados a este punto podemos concluir que hemos completado los objetivos planteados al principio de este proyecto. Entre las tareas desarrolladas, destacaríamos por su particular complejidad el caso de estudio real que hemos realizado a través de Feebbo, y que nos ha servido como experiencia a lo largo del desarrollo del proyecto. Llevarla a cabo ha traído consigo algunos contratiempos. Puede que el mayor de ellos haya sido la parte de investigación sobre cómo realizar encuestas para miles de personas con el objetivo de obtener respuestas que nos sirvan para nuestro caso de estudio y nuestra aplicación. Además, a dichas encuestas había que añadirle preguntas de personalidad que también tuvimos que investigar. Más aún, los resultados de las encuestas no eran inmediatas ya que queríamos alcanzar un número de encuestados bastante alto.

Como dato final, comentamos el precio y algunas características del móvil que obtiene alguno de los algoritmos (en este caso utilizamos el Algoritmo Genético) de nuestra aplicación al realizar la ejecución del caso de estudio real. El precio del móvil es de 154 euros y tiene las siguientes características: tarjeta Nano SIM, radio FM, bluetooth, resistente al agua, tiene tecnología NFC, móvil con Dual SIM y diseño juvenil.

Por último, destacar el potencial de la aplicación para el mundo laboral actual, en particular dentro del área del marketing.

11. Posible Trabajo Futuro

En este apartado explicaremos todas las cosas en las que podríamos trabajar en un futuro para mejorar y ampliar este proyecto.

Como ya hemos comentado además de la aplicación de escritorio, tenemos una aplicación móvil de este programa, si bien la app con alguna funcionalidad menos, como es el caso de lectura y escritura de fichero. Además, como es lógico los algoritmos se ejecutan mucho más lentamente en la app.

En un futuro podríamos hacer un sistema Cliente – Servidor en el que la app hiciera de interfaz para el usuario, para que elija los algoritmos que desee para realizar la ejecución, las variantes, introducción de datos manualmente, etc. Después de tener los datos necesarios para la ejecución, se lanzaría la petición al servidor, que se encargaría de realizar el algoritmo, ya que además lo haría mucho más rápido. Cuando la ejecución terminase, devolvería el resultado a la app para mostrárselo al usuario. Como idea, se podrían utilizar un sistema de servicios REST, ya que las transacciones mediante objetos Json facilitarían la comunicación con nuestro programa desarrollado en Java.

Otra de las mejoras posibles para el proyecto hace referencia a la obtención del precio óptimo para nuestro producto. Ahora mismo utilizamos una función de media ponderada, pero también valoramos otra opción que al final no se desarrolló. Esta opción consiste en estimar una función *lineal* que tenga tantos parámetros de entrada como características tenga el móvil y nos devuelva el precio. Dicha función se puede interpolar a partir de los precios de móviles reales que utilizaremos para modelizar la competencia, de forma que cada uno de estos móviles (junto con su precio conocido) es un punto de dicha función. Para estimar heurísticamente los coeficientes de la función lineal, podríamos utilizar un algoritmo genético: cada individuo (cromosoma) es una combinación concreta de pesos, y la función de fitness devuelve la distancia entre el precio real y el precio estimado con los pesos de ese cromosoma, descartando aquellos que tengan una distancia mayor, y quedándose con los que se ajusten más.

Otra posible futura mejora es añadir más algoritmos al programa para optar a una mayor variedad. Por ejemplo, se podría implementar el algoritmo Hill Climbing. Aunque se presupone peor que el Simulated Annealing, ya que se queda atascado en máximos locales, ofrece algunas variantes interesantes que podríamos estudiar.

Finalmente, podrían añadirse más variantes de los problemas a nuestro programa. Como ejemplo, podríamos desarrollar una variante de la versión con varios productos por productor en la que no tengan que tener todos los productores el mismo número de productos, y se pueda dar el caso de que un productor juegue con cuatro productos y otro solo con dos, o uno.

12. Plan de Proyecto

A continuación, explicaremos cómo nos hemos organizado y en qué medida hemos contribuido cada uno para el desarrollo del proyecto.

12.1 Organización del Proyecto

Hemos realizado reuniones cada dos o tres semanas con el director de nuestro proyecto Ismael, vía Skype o presencial. En cada una de las reuniones le comentábamos lo que habíamos hecho durante el transcurso de ese intervalo de tiempo. Además, le planteábamos las dudas que nos habían surgido y nuestra posible solución a esos problemas.

Por otro lado, antes de acabar cada reunión comentábamos con el director del proyecto lo que íbamos a hacer durante ese tiempo, de cara a la siguiente reunión.

A lo largo del desarrollo de nuestra aplicación, hemos utilizado las hojas de cálculo de *Google Drive* [32] para listar las tareas que teníamos que hacer y consecuentemente notificar cuando éstas estaban concluidas. Para que no haya confusión, decidimos que era mejor ir poniendo las fechas para cada tarea y por ende su duración. Hemos seguido la metodología *Scrum* [33]: marcamos en verde las tareas que han sido realizadas en el plazo que habíamos establecido, en amarillo las que se retrasaban y por último en rojo aquellas que no se cumplían o que no se llegaban a usar (por ejemplo, investigaciones que se hicieron en su debido momento para ver si podían servirnos en el proyecto).

16/10/2015	Primera Reunión Borja y Stephania		
19/10/2015	Estudio de investigación y código del programa original.		
29/10/2015	Reunión de Progreso	2 semanas para contactar con la Empresa PanelSilliker, el cual no tuvo éxito	
04/11/2015	Skype con Pablo: Resolver dudas técnicas		
07/11/2015	Información sobre la política de EEUU		
08/11/2015			
11/11/2015	Skype con Ismael: Planear el siguiente paso		
12/11/2015	Reunión para planear arquitectura		
15/11/2015	Investigar sobre estudios de mercado (feebbo, etc)	Creación repositorio en github y primero pasos con proyecto java.	
17/11/2015	Adaptación del proyecto a java. (Lectura de attrs. del excel)	Lectura de un excel usando la biblioteca Poi.	
18/11/2015	Merge de las ramas, reparto de trabajo.		
19/11/2015	Investigar sobre JGAP - Java Genetic Algorithms Package, para poder usar el algoritmo genético en el proyecto.	Reunión y un acuerdo con la empresa Feebbo para colaborar con nuestro proyecto de fin de grado.	
20/11/2015	Reunión con Ismael (clustering, tema y preguntas para las encuestas, proyecto de java)	Reparto de trabajo	Próximo paso hablar con la empresa Feebbo.
21/11/2015	Empezar a investigar sobre clustering.	Comienzo de la encuesta para entregarles a Feebbo	Investigación sobre móviles del mercado

Figura 32 Metodología Scrum

Además, hemos anotado todas las referencias a páginas que hemos utilizado para realizar nuestra investigación.

En cuanto a la organización que hemos utilizado al desarrollar el código, creímos conveniente y cómodo usar el sistema de control *Git* con un repositorio público en *GitHub* [34]. Allí hemos creado un repositorio con nuestro proyecto .git, que tenía una rama master y una rama para que cada uno pudiéramos trabajar en paralelo, y realizar las pruebas correspondientes.

Cuando ambos terminábamos de hacer las pruebas satisfactoriamente, actualizábamos la rama master para que tuviera siempre una versión estable.

12.2 Contribución al Proyecto

En este apartado ambos vamos explicar individualmente nuestra contribución al proyecto Marketing Computacional.

12.2.1 Stephania Cristina Hinostroza Hualpa

En este apartado voy a describir mi aportación individual y mencionar las dificultades que me han surgido a lo largo del desarrollo del proyecto.

Para empezar con el proyecto realizamos un estudio del trabajo inicial proporcionado por el profesor para poder incorporarlo en el proyecto. Se investigó sobre los diferentes algoritmos que utilizaban, y decidir finalmente si seguiríamos con el mismo lenguaje de programación o por lo contrario optimizar la implementación con otro lenguaje.

Después de decidir el lenguaje de programación y la herramienta que utilizaríamos para este proyecto: Java y Eclipse (Android Studio para la app de móvil) respectivamente, tuvimos que documentarnos sobre la forma de llevar a cabo la implementación y optimización de los algoritmos. En primer lugar, me documenté sobre las diferentes librerías que nos permitían facilitar el uso de esos algoritmos, por ejemplo: librería Poi para la lectura de ficheros Excel, JGAP (Java Genetic Algorithms Package) que contenía la implementación del algoritmo genético, Java CSV para la lectura de ficheros CSV, etc.

De todas las librerías que he investigado, algunas se han utilizado durante el proyecto mientras que otras se han ido descartando a lo largo del desarrollo. Por ejemplo, la librería Poi se descartó porque nos producía problemas al leer el fichero Excel del caso de estudio abordado en [2] sobre la política de España, y la librería JGAP

se descartó también porque decidimos implementar el algoritmo genético, basándonos en lo que nos habían proporcionado.

En paralelo a esa investigación, realicé otra sobre empresas que hacían estudios de mercado. Como partía de un breve conocimiento de alguna de estas empresas, decidí profundizar en el tema y ponerme en contacto con alguna de ellas para saber si podrían ayudarnos de alguna manera y de forma desinteresada. Como nuestra idea era hacer un caso de estudio real sobre un determinado producto, lo que queríamos era publicar una encuesta para que lo pueda realizar mucha gente, y con esos resultados poder generar nuestro caso de estudio. Me puse en contacto con una empresa, pero finalmente no obtuvimos respuesta, decidí seguir intentando y con la segunda empresa con la que contactamos, esta vez a través del profesor por correo, recibimos una buena respuesta por parte de la empresa Feebbo, ya que colaboraban con diferentes universidades a nivel internacional. Esta se dedica a realizar estudios de mercado a través de encuestas realizadas por sus clientes, y a vender esos resultados a otras empresas interesadas en diferentes productos. Tuvimos diversas reuniones con el director de Feebbo, Goyo Hernández, para acordar cómo nos ayudarían y cuál sería nuestra colaboración con ellos, sin tener que pagar por los resultados de la encuesta. Finalmente, el acuerdo fue que podíamos usar su plataforma para subir nuestra encuesta y que podamos utilizar a sus clientes para que respondieran a las preguntas de la encuesta, a cambio de que les ayudásemos a clasificar a sus encuestados por perfiles de similitud.

Para realizar lo que nos pedía Feebbo, tuve que documentarme sobre algoritmos de agrupamiento y verificar cuál nos podría servir conforme a lo que nos pedían. Para utilizar estos algoritmos, decidí que la herramienta Weka me podía servir para hacer pruebas sobre clusterización. Elegí esta herramienta porque ya la había utilizado en una asignatura en la carrera, e inmediatamente recordé que podía clasificar los perfiles en clusters agrupados por similitud, aprovechando los diversos algoritmos que me proporcionaba Weka. Como no recordaba todas las funcionalidades de Weka, me puse a investigar más en profundidad sobre la herramienta. El algoritmo que elegí fue K-Means, ya que después de investigar se adecuaba a lo que nos pedía Goyo.

Además, esta herramienta tiene una librería Weka.jar que nos permitía utilizar el algoritmo en Java. Realicé una implementación en Java sobre el algoritmo K-Means (crear y configurar el algoritmo, cargar los datasets, entrenar el algoritmo de clusterización, e identificar los clusters de cada instancia) y finalmente mostrar los centroides utilizando la librería de Weka. Después de que realizara la implementación y las pruebas pertinentes tanto en la herramienta Weka como en Java, tuvimos una reunión con Goyo en el que le expliqué desde cero sobre los algoritmos de agrupamiento, sobre la herramienta Weka (uso, licencia, etc.), el algoritmo utilizado, y

las pruebas básicas de las dos formas. Se le proporcionó toda la información que necesitaba, así como la implementación que hice en Java.

La implementación que realicé del algoritmo de clusterización también nos sirvió para añadirlo como variante en nuestro proyecto mencionado en apartados anteriores, esta variante nos permite tener resultados mejores.

Fui la encargada de realizar la interfaz para escritorio desde cero y de añadirle todas las funcionalidades posteriormente. Utilicé la herramienta Netbeans para tener una idea conceptual de la interfaz para escritorio que posteriormente implementaría en Eclipse, desarrollé una pequeña implementación que fue mejorando conforme avanzaba el proyecto.

Ademas de tener ya implementados esos algoritmos, había que estimar el precio del producto, y para ello tuve que investigar sobre algoritmos de interpolación y herramientas que podríamos utilizar. Así que durante la investigación encontré dos herramientas que a priori parecía que nos iba a ser útil: MatLab y ArcGis, pero después de trastear un poco con estas herramientas y hacer diversas pruebas, decidí descartarlas porque no hacían realmente lo que queríamos. Finalmente se optó por implementarlo por nuestra cuenta.

Se propusieron dos nuevos algoritmos: Algoritmo de Optimización por Enjambre de Partículas y Algoritmo de Enfriamiento Simulado. Para saber un poco más de estos algoritmos (ya que no lo habíamos visto en la carrera) me puse a buscar información útil que nos sirviera tanto para comprender los algoritmos como para saber implementarlos. Encontré diversos ejemplos básicos que me hicieran entender la funcionalidad y el comportamiento de estos algoritmos, así como esquemas generales en pseudocódigo que nos sirvieron de ayuda a la hora de implementar. Al adaptar dichos algoritmos en Eclipse, también iba actualizando y mejorando la interfaz gráfica de escritorio para visualizar los resultados, así como solucionando los fallos que se producían.

En cuanto a las abstracciones de estos algoritmos, estaba encargada de organizar el código para poder optimizarlo. Implementé una clase donde se encontraban todos los métodos y variables comunes que hay en cada algoritmo, con la finalidad de no duplicar código, sino optimizarlo de la mejor forma posible.

Las variantes del proyecto que se iban implementando también se adaptaban para la interfaz de escritorio (la variante de atributos enlazados tuve que implementarlo para que se pudieran introducir a través de la interfaz), ya que nos permitía introducir los datos de entrada (tuve que realizar la implementación para la creación de los atributos, productores y perfiles) y modificar diferentes variables propias de cada algoritmo y las variables comunes entre dichos algoritmos para hacer las pruebas necesarias. Esta interfaz también permitía leer de un fichero txt y xml, para

ello generé un formato específico para cada tipo de fichero que tendrían que tener todos los archivos para que pueda ser leído a través de la interfaz. Después de crear el formato, implementé la lectura para los dos tipos de ficheros, y también la escritura para el caso en el que se introduzcan los datos por la interfaz y se quiera almacenar esos datos para futuras pruebas. Ademas añadí un método que nos permita almacenar los resultados de cada implementación en un documento de texto para ser visualizadas cuando se necesiten. Incorporé la visualización de los resultados de la clusterización en la interfaz, así como las listas que son generadas al realizar una ejecución de cualquier algoritmo.

La ventaja de leer de estos tipos de ficheros es que se pueden hacer diferentes pruebas y sobre todo me sirvió para pasar los resultados de la encuesta a nuestro formato. Feebbo nos proporcionó los resultados de las encuestas en diferentes ficheros csv, el cual tuve que organizar y clasificar perfiles para poder transformarlos en nuestro formato y poder realizar las ejecuciones del caso de estudio. Cómo eran más de 800 encuestados, se hizo un poco complicado organizar dicha información, ya que estaba estructurada a través de códigos, lo cual hacía que me fuese más tedioso realizar la transformación.

Finalmente, realizamos las pruebas con los resultados finales, probando diversas combinaciones de variantes y con muchas ejecuciones para poder hacer los gráficos mostrados en apartados anteriores.

En cuanto a los errores que se han producido durante el desarrollo del proyecto, ambos hemos solucionado estos problemas dependiendo de nuestra carga de trabajo y disponibilidad, ya que cada uno tenía unas tareas asignadas. Mientras mi compañero realizaba alguna implementación, yo organizaba, verificaba y documentaba la memoria para reflejar lo que hasta el momento habíamos hecho y utilizado, y a la vez que modificaba los cambios necesarios que el tutor indicaba tras cada reunión.

También añadí a la memoria diagramas de clases sobre las clases más relevantes del proyecto a través de un plugin de Java llamado ObjectAid UML Explorer, el cual descubrí investigando y haciendo pruebas, ya que descarté utilizar ArgoUML después de haber realizado muchas pruebas y no llegar a obtener el resultado que quería.

Después de la investigación que realizó mi compañero sobre los móviles de la competencia, organicé la información de esos móviles en un documento con tablas descriptivas para saber sus características en correspondencia con los que se preguntaba en la encuesta.

Debido a circunstancias de disponibilidad, he tenido más comunicación con el tutor en las reuniones para comentarle sobre los avances de las semanas.

12.2.2 Borja Salazar Rey

En este apartado explicaré brevemente lo que tuve que hacer para la realización de este proyecto, y las dificultades encontradas en el camino.

En primera instancia, lo que tuve que hacer fue documentarme sobre que lo habían hecho anteriormente Ismael y Pablo. Ellos tenían ya un proyecto comenzado en el que nosotros nos íbamos a basar, así que durante unas semanas estuvimos, tanto Stephania como yo, aprendiéndonos y leyéndonos toda su documentación para poder tener una base sobre la que empezar nuestro proyecto.

Lo primero que pensamos es que necesitábamos un caso de uso real. Ismael Rodríguez y Pablo Rabanal utilizaron el tema de la política, pero nosotros queríamos orientarlo más hacia un producto de mercado. Aquí contactamos con la empresa Feebbo. Después de tener un acuerdo con ellos, fui yo el encargado de mantener el contacto con ellos. Lo primero que hubo que hacer fue crear una encuesta para colgarla en su web y así sacar todos los datos que nosotros necesitábamos. Durante una temporada estudié el mercado actual de Smartphones (hacia lo cual queríamos orientar el caso de uso), para así componer una encuesta que nos proporcionara los datos suficientes para poder crear un móvil con los mayores detalles posibles.

Después de tener esta encuesta, los contactos de Feebbo nos aconsejaron que, si queríamos que la gente no se cansara de hacer nuestra encuesta y así tener un número mayor de encuestados, deberíamos partirla en tres encuestas enlazadas, y, además, al final de cada una de ellas poner preguntas sobre personalidad, ya que el eslogan de la encuesta sería “Te decimos cómo eres según el Smartphone que te gusta”. De este modo, tuve que buscar tests de personalidad con sus respectivas respuestas, e introducir esas preguntas en nuestras encuestas para poder ofrecerle al usuario de Feebbo una experiencia más completa.

A partir de ahí, mantuve el contacto con Feebbo para conocer el estado de la encuesta, y para que nos fueran proporcionando los datos necesarios para nuestro proyecto.

Paralelamente a esto, comenzamos el proyecto en base al código de Pablo Rabanal. Tanto el Algoritmo Genético como el Minimax fueron adaptados a Java entre Stephania y yo, pero aún nos daban problemas. Más adelante, me encargué de solucionar dichos errores en los algoritmos para hacerlos funcionar definitivamente.

Después de esto, separamos dos versiones del programa, la aplicación Android y la aplicación de escritorio. En mi caso, fui el encargado de hacer el desarrollo de Android, junto con toda la interfaz de usuario.

Más adelante se decidió que, aparte de introducir los datos aleatoriamente el usuario debería poder introducirlos a mano, y nos vimos obligados a crear una interfaz más compleja para poder darle este servicio al usuario. Estas pantallas llevaron más trabajo del esperado, ya que en la versión móvil se crean los formularios de introducción de datos dinámicamente, y puede dar muchos problemas.

Además de eso, pensamos en que el usuario pudiera añadir los datos mediante un fichero de texto. Como yo era el responsable de la parte de Android, me puse a investigar cómo se podría hacer, y tras varias semanas investigando no conseguí hacer que funcionase, dado que la idea era que el usuario pudiera explorar todos los archivos de su móvil y elegir de entre cualquiera de ellos el que él quería para la instancia del problema. Si el archivo no fuera válido se le comunicaría. Pero solo se consiguió con rutas estáticas que apuntaran a una ruta dentro del mismo paquete de la aplicación, cosa que no nos servía para nuestro objetivo, puesto que un archivo guardado de esta forma nunca se podrá cambiar y nos haría imposible cambiar la instancia para ejecutar. Esta idea se descartó para la versión de Android, ya que estaba consumiendo demasiado tiempo.

También fui el encargado de hacer la implementación de la interpolación, para obtener el precio de nuestros posibles productos. Barajamos varias opciones para ello, como se explica anteriormente. Fui el encargado de hacer la primera versión de la interpolación mediante la media ponderada, y cuando fuimos a hacer la interpolación basada en un algoritmo genético, descartamos la idea por falta de tiempo, y porque había otras cosas con mayor prioridad.

Siendo capaces que obtener el precio de los productos, fue el momento de comenzar a implementar las variantes de los problemas. Las tres variantes elegidas fueron: obtener el producto con el que más beneficios se pudieran conseguir, hacer que la puntuación final de un producto pudiera depender de combinaciones de valores de atributos (atributos Linkados) y, por último, que cada productor pueda tener más de un producto en el mercado. Fui el encargado de implementar estas variantes. La mayoría de estas variantes se centraban en el cambio de la función de Fitness, haciendo que contase con unas cosas u otras, o que sumasen más o menos dependiendo de ciertas circunstancias. Estos desarrollos se hicieron en la versión de la aplicación de Android, y luego fueron entregados a Stephania para que los introdujera en la versión de escritorio.

Más adelante se decidió implementar nuevos algoritmos aparte del Genético y el Minimax que desarrollaron Ismael y Pablo. Nos aconsejaron el PSO y SA. Gracias a la información que me proporciono Stephania sobre estos algoritmos, realicé el desarrollo de estas nuevas maneras de resolver nuestro problema. Después de acabarlas, se añadieron las variantes que anteriormente había hecho para los otros dos algoritmos. Estos desarrollos ocuparon varias semanas de desarrollo, dado que era

algo completamente nuevo, de lo que no teníamos ningún referente cercano como en el Genético o el Minimax.

Por último, menciono el trabajo relacionado con la abstracción de los algoritmos. Había que conseguir que los algoritmos fueran independientes del problema que están resolviendo y que la implementación del problema no dependiera de con que algoritmo está siendo resuelto. Esto fue un esfuerzo extra, porque para abstraerlo ahora había que hacerlo para cuatro algoritmos distintos, y teniendo en cuenta las variantes para cada uno de ellos.

En la parte de la memoria, contribuí en los puntos de Implementación, para explicar cómo funcionan los algoritmos y cada una de sus clases, y retoqué alguna cosa referente al diseño del proyecto. Además, contribuí en el apartado de Conclusiones, y en los resultados finales de la aplicación. Durante todo el desarrollo de la memoria estuve pendiente de su evolución.

13 Bibliografía

[1] «Feebbo,» [En línea]. Available: http://www.feebbo.com/es/.

[2] Ismael Rodríguez, Pablo Rabanal, Fernando Rubio, «How to make a best-seller: optimal product design problems,» Enviado para su posible publicación 2016.

[3] «Consumolab,» [En línea]. Available: http://www.consumolab.es/.

[4] «PanelSilliker,» [En línea]. Available: https://www.panelsilliker.com/.

[5] «MySurvey,» [En línea]. Available: http://es.mysurvey.com.

[6] «SurveyMonkey,» [En línea]. Available: https://es.surveymonkey.com/.

[7] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico.

[8] «Introducción a los algoritmos genéticos y sus aplicaciones,» [En línea]. Available: http://www.uv.es/asepuma/X/J24C.pdf.

[9] «JGAP,» [En línea]. Available: http://jgap.sourceforge.net/.

[10] «Algoritmo Minimax,» [En línea]. Available: https://es.wikipedia.org/wiki/Minimax.

[11] «Interpolación,» [En línea]. Available: http://www.uv.es/diazj/cn_tema5.pdf.

[12] J. V. A. P. L. Matencio, «Ajuste e Interpolación Multivariable de Funciones,» [En línea]. Available: http://campus.usal.es/~3cm/sites/default/files/3CM-18.pdf.

[13] «ArcGIS Spatial Analyst,» [En línea]. Available: http://ficheros.esri.es/archivos/documentostecnicos/ArcGIS_Spatial_Analyst_anexo_producto.pdf.

[14] «MatLab,» [En línea]. Available: http://es.mathworks.com/index.html?s_tid=gn_logo.

[15] «Conjunto de Herramientas de Interpolación,» [En línea]. Available: http://help.arcgis.com/es/arcgisdesktop/10.0/help/index.html#/na/009z00000069000000/.

[16] «MathWorks: Interpolation,» [En línea]. Available: http://es.mathworks.com/help/matlab/interpolation-1.html.

[17] «Comparación de diferentes algoritmos de clustering,» [En línea]. Available: http://www.sc.ehu.es/jiwdocoj/remis/docs/GarreAdis05.pdf.

[18] «Algoritmos de agrupamiento,» [En línea]. Available: http://marmota.dlsi.uji.es/WebBIB/papers/2007/1_Pascual-MIA-2007.pdf.

[19] P. R. Mark Hall, «WEKA KnowledgeFlow Tutorial,» 2008 July 14. [En línea]. Available: http://software.ucv.ro/~eganea/AIR/KnowledgeFlowTutorial-3-5-8.pdf.

[20] «More Data Mining with Weka (Representing clusters),» Weka, [En línea]. Available: https://www.youtube.com/watch?v=HCA0Z9kL7Hg.

[21] «Uso básico del API de WEKA,» [En línea]. Available: http://ccia.ei.uvigo.es/docencia/MRA/practicas0809/api-weka.html.

[22] «Weka: The University of Waikato,» [En línea]. Available: http://www.cs.waikato.ac.nz/ml/weka/.

[23] «Weka (aprendizaje automático),» [En línea]. Available: https://es.wikipedia.org/wiki/Weka_(aprendizaje_autom%C3%A1tico).

[24] «Use WEKA in your Java code,» [En línea]. Available: https://weka.wikispaces.com/Use+WEKA+in+your+Java+code.

[25] «Index of /Datasets/UCI/arff/,» [En línea]. Available: http://repository.seasr.org/Datasets/UCI/arff/.

[26] «Apple,» [En línea]. Available: http://www.apple.com/es/.

[27] «Samsung,» [En línea]. Available: http://www.samsung.com/es/home/.

[28] «MediaMarkt: Telefonía Móvil,» [En línea]. Available: http://tiendas.mediamarkt.es/telefonia?gclid=Cj0KEQiAycCyBRDss-D2yIWd_tgBEiQAL-9Rkl2WjUb3SY-KwS36V0PDU6W5AsiRWwD5XysBRq_GUqQaAiBl8P8HAQ.

[29] «Fnac: Telefonía Móvil,» [En línea]. Available: http://www.fnac.es/telefono-MP3-GPS/Smartphones-Libres/s39887#.

[30] «PC Componentes: Telefonía Móvil,» [En línea]. Available: http://www.pccomponentes.com/smartphones_gps.html.

[31] «Ubergizmo,» [En línea]. Available: http://www.ubergizmo.com/.

[32] «Google Drive,» [En línea]. Available: https://www.google.com/intl/es_es/drive/.

[33] K. S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile, Upper Saddle River, NJ : Addison-Wesley, 2012.

[34] «GitHub,» [En línea]. Available: https://github.com/.

[35] «The Apache POI Project,» [En línea]. Available: https://poi.apache.org/.

[36] J. Bell, Machine Learning: Hands-On for Developers and Technical Professionals, Wiley.

[37] L. d. S. Coelho, A quantum particle swarm optimizer with chaotic mutation operator, vol. 37, Chaos Solitons & Fractals, 2008, pp. 2-22.

14 Summary in English

14.1 Introduction

In this first introductory section we will describe briefly how the report will be structured. In addition, we will explain the objectives we have raised for the project Computational Marketing: Automatic Product Design.

14.2 Project Organization

The report is structured as follows:

- In the first chapter, the project goals are exposed.
- The second chapter describes the problems addressed the application.
- The third chapter will discuss the state of art. We will mention the activities developed by several existing companies, related to market research.
- The fourth chapter presents how we carried out the design of our application.
- In the fifth chapter, we describe the functionality of our desktop and Android applications.
- In the sixth chapter, we will mention what tools we chose for the implementation and why they suit our necessities.
- In the seventh chapter, the implementation of the multiplatform application will be discussed.
- In the eighth chapter we describe the surveys we designed to be conducted in the Feebbo platform, as well as the type of questions asked and the results obtained.
- In the ninth chapter we will present the results achieved in our case study of the application.
- In the tenth chapter we give the final conclusions of this project. We will also discuss possible extensions and improvements.
- In the eleventh chapter will discuss the future work, describing the extensions that can be made to the project.
- In the twelfth chapter we will present the organization of the project development, as well as and the contribution of each member of the group.
- The thirteenth chapter is the bibliography.
- The fourteenth chapter provides a summary of the objectives, conclusions and future work in English.

14.3 Goals

The main objectives of the project are:

1. Develop an application that, based on the customer preferences about the characteristics of a specific product, and taking into account the products offered by competitors, obtains the ideal product to be sold immediately or in the long term, that is:
    a) Design a product completely from scratch to maximize the number of customers or to reach a certain number of customers; or
    b) Design a strategy indicating how to modify our product as the competitors modify their own, so that we reach some average number of customers within a specified period regardless of changes that have made the competitors.
    It is also considered an alternative version of the above problem in which the deadline is restricted: the number of turns cannot be greater than the number of product attributes.

Our application will solve several variants of such problems, and it will allow to apply different algorithms. It will also offer both a desktop version for Windows as a mobile version for Android.

2. We will demonstrate the usefulness of the application carrying out a realistic full case study.
3. Finally, we will compare various algorithms in the resolution of various variants of the problems.

In order to achieve this purpose, we need to develop the following activities:

- Deepen our knowledge on genetic algorithms, algorithms particle swarm optimization, simulated annealing algorithms, minimax algorithms and interpolation algorithms, to apply them in the development of our application and our case study.
- Design and carry out a set of questions for surveys allowing us to, get opinions of respondents' preferences on a particular product (in our case mobile phones) through the website Feebbo [1].
- Learn different clustering algorithms to classify respondents in clusters organized by similarity.
- Design our application, choose the tools to be used in order to develop a multiplatform system.
- Develop and implement the application that designs the best product.

- Improve the ability of programming in the Java language and learn to use functions contained in its libraries.
- Evaluate the results of the implementation and analyze them in terms of the obtained products.
- Elaborate documentation collecting everything we have done and enable the future expansion of the project.

14.4 Conclusions

In this project we have implemented algorithms to solve problems PDO, SPDG and lSPDG, three problems trying to encapsulate essential objectives in marketing: selecting a product that achieves a high number of clients immediately (PDO), and selection of a strategy of sustainable product design that achieves a certain number of customers in a long or short term (in SPDG and lSPDG respectively). To solve these problems, we used sub-optimal heuristic methods, instead of using optimal algorithms that would be intractable.

We developed experiments with random data and data from a real case study about mobile phones.

For conducting the objectives that we raised we have designed a multiplatform interface so that the results can be displayed clearly and the user can enter data through the application. Furthermore, as to the implementation of the application, the code has been abstracted and well organized so that it can be reusable and can be added different algorithms with new problems.

Considering the three goals we proposed at the beginning of the development of this project, we have developed a useful and efficient program that allows us to solve our main objective: decide the characteristics of the product to sell to maximize the number of clients immediately or long term. We have conducted a real case study to apply our tool.

In order to complete the third goal of the project, next we analyze the experimental results described in the previous section, and we present our conclusions about the algorithms we have implemented.

We begin with the PDO problem and genetic algorithm. This algorithm has a fairly stable initial results. Although in some cases we get a result that departs from the average, it is in the minor occasions, and it seems to depend a lot on the instance with which we execute the algorithm. Final results usually reach between 40 and 45 percentage points, thus obtain stable end results, but are among the lowest of all algorithms. In addition, the rate of improvement on the results of the initial population

of the algorithm is also the lowest of all algorithms, only an average improvement of 12 percentage points per execution.

In the PSO algorithm we have observed curious facts. For example, the initial product of this algorithm begins taking 17% of the initial customers, one of the initial lowest situations, something strange as both the GA and the PSO get their initial situation in the same way and should have initial scores similar. This makes us think there might be a bug in the code which has not been found, as the two algorithms generate the population in the same way, and one begins with a 17 % market share, and another with 35%. We cannot discard the idea of a bug even in spite of the fact that, after studying the implementation of step with small algorithm instances (enough to keep track manually), we found that the algorithm works correctly.

Finally, we analyze the results of simulated annealing algorithm. This results of this algorithm are normal, without any abnormality. Initial results match the average results obtained by all other algorithms and so does their final results. As the results, It is neither the algorithm with the worst beginning nor the algorithm with the best ending, but always stays between 45 percentage points, so we can say that it is a stable algorithm.

For the problem SPDG and minimax algorithm, the percentage of customers achieved in the initial situation is low, around 10%, and yet the minimax algorithm obtains a product optimization, achieving reach about 55% of market customers. Keep in mind that in this problem the competing product is also improved, so it is also the most realistic, since it is difficult to believe that a competitor will not make his product evolve to get more benefits than your opponent.

This algorithm is very stable, having a minimum difference of 0.9 percentage points from its best and its worst result.

Since the Minimax and GA, PSO and SA solve various problems it is useless to compare their results. The Minimax trying to solve a problem SPDG, while others solve the problem PDO. Furthermore, these algorithms are using different instances, so definitely comparing them is pointless, as they would not provide any useful information.

Now let's analyze the results of these algorithms with data from our own case study. Let us remember that this instance is constructed from the results of surveys that ask the company Feebbo. Discuss the major discrepancies regarding the case studies of random instances.

For example, one of the things that draws our attention is that, except in the minimax, all other algorithms improve considerably (by 15 percentage points) their initial and final situations.

In the case of Minimax, it is the other way around, it worsens their final results compared to the case study with random instances, but it is curious that the improvement is the same. In this case, it starts 4 percentage points lower and ends the same, with the final score 4 points below the random case study. This means that the improvement is the same, indicating that the algorithm is very stable, and this drop in the initial results may simply be due to the different instance, since the amount of data handled is significantly different (as well as the data themselves).

The results of the algorithm SA are the same. While we have said that their initial and final result increases, we must stress that it increases the same as between random instances. And the rate of improvement between the case of randomized and real case study is the same, 17 percentage points. Similarly, as in the case of the Minimax, this indicates that the algorithm is stable and this difference may be due to significant changes of the instance used.

For GA and PSO, something remarkable and unsual happens. Both algorithms give similar results, which is not surprising because of their resemblance. However, these results are somewhat unusual. While both algorithms start with a higher initial situation compared to the case of random instances, both get a strangely high final result, as both algorithms exceed 90 percentage points (in the case of GA, it goes up to 96). We do not have a certain explanation for this increase of efficiency of the algorithm, but given that this increase affects both algorithms almost in the same way we can probably discard an error in the execution of the algorithm, since it is not likely that two bugs, one at each algorithm, affect the results of both algorithms in the same way, and that's complicated. Comparing both algorithms, once again the PSO optimization is higher. The abnormally high performance of both algorithms could be due to the particular instance under use, which in turn was built from the replies collected from our Feebbo surveys. In particular, our way to convert data from these surveys into input data for the PDO problem could have not given rise to an instance that faithfully captures the difficulty of dealing with competitors, or to appropriately combine the attributes of products. As a result, for the obtained instance, it is surprisingly easy to grasp a huge amount of market (it is hard to believe that, if there really a huge unsatisfied demand, manufacturing mobile phones companies had not realized yet).

Next we analyze the results of different algorithms using clustering. The first one we discuss is the GA. Comparing both cases (i.e. using or not clustering to generate the initial population), the number of customers reached by the best individual of the initial configuration is 10% higher in the case of clustering. Regarding the customers rate achieved at the end of the algorithm, the rate of the algorithm with clustering is also higher than the rate of the algorithm without clustering.

With the PSO algorithm we observe the same behavior as for the GA. The rate of improvement is the same in both cases, and when we use clustering, we managed to start the algorithm in a more advantageous position than without it. The final results are also better.

We consider the results for variants that maximize the benefits as opposed to customers. For PDO, we can see that the PSO and SA algorithm obtain an increase of income like of around 150 percentage improvement, whereas the GA reaches more than 250 points.

In this version of the problem, the algorithm that seems get better results is the genetic with an improvement of 275 percentage points. We do not really know why the PSO algorithm gets much worse results in this variant of the problem if executions for standard PDO do not take much difference.

With respect to the results obtained in the resolution of the variant in which each producer can sell simultaneously with several products, we observe the following distinctions in the three algorithms applied to this variant.

If we add up the obtained percentage points achieved by the three products sold by each producer, it gives us an initial situation according to the data obtained in standard executions.

With respect the final results, it happens the same. The obtained percentage points achieved by products are lower than in the standard execution, but if we add those obtained by the three products sold together, we get a higher score than the normal version. This is because the genetic algorithm is applied to three different products jointly. If we are able to optimize a single product to some extent, and apply the same technique to two other products in a coordinated way, we get better results, as we will be obtaining better products, which are coordinated to efficiently divide the demand space.

In the last experiment, which corresponds to the execution of Minimax with different depths, it is clearly seen that when we run the problem with a depth 2 for the opponent and 4 for us, we get a final score of 53 percentage points. However, if we execute the algorithm with superior depth for us (2 for the opponent, 8 for us), we obtain a better final score, 57 percentage points. So we see that, if we reduce our depth and expand that of our opponent, the results will be worse for us and better for the opponent. Therefore, the result obtained by the Minimax depends on the depth to which we are willing to go. We should not forget that the running time grows exponentially with the depth.

Finally, at this point we can conclude that we have completed the goals set at the beginning of this project. Among the tasks performed, we highlight the particular

complexity of the real case study we developed with the aid of Feebbo. It has served as an experience throughout the project development. Carrying it out has brought some setbacks. The greatest of them may have been the part of research on how to conduct surveys for thousands of people in order to get answers that help us to our case study and our application. Moreover, in these surveys we had to add questions of personality which we also had to investigate. In addition, the survey results were not available immediately, as we wanted to reach a fairly high number of surveyed people.

As final data, we present the price and some of the features of the mobile phone that gets one of the algorithms of our application to perform the execution of real case study (in this case we used the Genetic Algorithm). The price of the mobile phone is 154 euros and has the following features: Nano SIM card, FM radio, Bluetooth, waterproof, have NFC technology, mobile with Dual SIM, a young design.

Finally, we spotlight the potential application for today's workplace, particularly in the area of marketing.

14.5 Possible Future Work

This section we will explain all the tasks we could carry out in the future to improve and expand this project.

As we have already mentioned, in addition to the desktop application we have also developed a mobile application of this program, even though without some functionalities, such as the reading case and writing to file. Moreover, as it is expected the algorithms execute much slower in the mobile app.

In the future we could make a Client – Server system in which the app would be just the interface for the user to choose the algorithms you want to execute, variants, manual data entry, etc. After having the necessary data for the execution, the request would be sent to the server, which would then carry out the algorithm, as it would run it much faster. When the execution would finish, it would return the result to the app to show it to the user. As an idea, you could use a REST system services as transactions through Json objects to facilitate the communication with our developed program in Java.

Another possible improvement for the project refers to obtaining the optimum price for our product. Right now we use a weighted average function, but we also considered another option that we finally did not develop. This option is to estimate a linear function having as many input parameters as features has the mobile phone, which would return us the price. This function can be interpolated from the actual

prices of the mobile phones we use to model the competition, so that each of these phones (along with its known price) is a point of that function. To heuristically estimate the coefficients of the linear function, we could use a genetic algorithm: each individual (chromosome) is a specific combination of weights (coefficients), and the fitness function returns the distance between the actual price and the estimated price if we use the weights of that cromosome. The GA would discard those that have a greater distance and would keep the ones which fit more.

Another possible future improvement is to add more algorithms. For example, we could implement the Hill Climbing. Although algorithm it is assumed that it is worse than Simulated Annealing since it gets stuck in local maximums, it offers some interesting variations that we could study.

Finally, more problem variants could be added to our program. As an example, we could develop a variant of the version with different products for each producer, where each producer may have a different number of products. For instance, a user could play with four different products whereas another user with only two. The number of products would not have to be the same for all producers.