UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

Diseño de Metaheurística para problemas combinatorios costosos

Autor
Irene Trigueros Lorca

Directores
Daniel Molina Cabrera
Francisco Herrera Triguero

Facultad de Ciencias

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Facultad de Ciencias

Granada, Junio de 2023

UNIVERSIDAD DE GRANADA

Diseño de Metaheurística para problemas combinatorios costosos

Autor
Irene Trigueros Lorca

Directores
Daniel Molina Cabrera
Francisco Herrera Triguero

Diseño de Metaheurística para problemas combinatorios costosos

Irene Trigueros Lorca

Palabras clave: metaheurística, combinatorio, costoso, optimización, algoritmo genético, exploración, explotación, diversidad

Resumen

Existen problemas de mucho interés, no solo teórico, sino también que poseen aplicaciones significativas en el mundo real, cuyo coste de evaluación es excesivamente elevado, o incluso inasequibles (a este tipo de problemas los llamaremos problemas costosos o problemas *expensive*), como podrían ser los problemas de optimización de redes neuronales, por lo que resulta interesante desarrollar algoritmos capaces de obtener soluciones competitivas en muy pocas evaluaciones. Además, en la literatura ya es posible encontrar algoritmos específicos en problemas *expensives* de optimización continua; pero no ocurre lo mismo con el caso de problemas combinatorios, para los que no se han diseñado algoritmos específicamente para este tipo de problemas, pero que son importantes de tratar ya que es cada vez más común usar problemas combinatorios en problemas complejos.

Por lo tanto, en este proyecto se propone un algoritmo metaheurístico nuevo especialmente diseñado para la resolución problemas combinatorios costosos (*expensive*), siguiendo un formato semejante al de un diario de desarrollo. El objetivo principal de este algoritmo será encontrar buenas soluciones en una reducida cantidad de tiempo.

Inicialmente se presentará un contexto matemático necesario para entender cómo resolver problemas de minimización sin restricciones y el cómo y porqué usar tests estadísticos (y cuáles usar) para la comparación de los distintos algoritmos.

Tras esto, se sigue la exposición de los algoritmos genéticos clásicos que se han usado como base para el diseño del algoritmo a desarrollar y las modificaciones que se le introducirán a estos conforme se avance en el desarrollo del algoritmo; ambas contendrán pseudocódigo asociado para su más fácil interpretación.

El diseño del algoritmo objetivo se hará como se indica a continuación. En primer lugar, se tomará como algoritmo base un algoritmo genético y, una vez analizado su rendimiento, se implementarán una serie de modificaciones que, progresivamente, compondrán la versión final del algoritmo. Estas modificaciones se justificarán observando y analizando los resultados obtenidos en los intentos anteriores con el objetivo de encontrar formas de aprovechar al máximo las pocas evaluaciones que nos podemos permitir en este tipo de problemas. Se describen detalladamente las tareas adicionales llevadas a cabo propias del desarrollo de un algoritmo de esta clase. Además, se realizarán análisis experimentales que irán demostrando que las sucesivas versiones del algoritmo van mejorando las anteriores.

Como no existen algoritmos específicos para la resolución de problemas combinatorios costosos, no podemos comparar los resultados finales obtenidos con ningún tipo de *benchmark* con el fin de obtener unas conclusiones. Por ello, el análisis de la potencia del algoritmo desarrollado se basará en la comparación con los resultados de las anteriores versiones desarrolladas y un análisis del porcentaje de mejora con respecto algoritmo genético clásico que se ha usado como base. Las conclusiones que se alcanzan indican que el algoritmo presentado en este proyecto es altamente competitivo.

Metaheuristics design for expensive combinatorial problems

Irene Trigueros Lorca

Keywords: metaheuristic, combinatory, expensive, optimization, genetic algorithm, exploration, explotation, diversity

Abstract

There are very interesting problems, not only in the theoretical field, but they also have significants applications in real life, which their computational cost is excessively high, or even unaffordable (these kind of problems will be refered as expensive problems), such as the neural network optimization problems, therefore it is compelling to develope algorithms that are able to obtain competitive solutions in few evaluations. Furthermore, specific algorithms for the optimization of expensive continous problems can be found in the literature; but that is not the case for the combinatory expensive problems, for those which no algorithms able to solve them have been developed, though they are important to deal with as is increasingly common to use combinatorial problems in complex problems.

Hence, throughout this project, an original metaheuristic algorithm is going to be proposed with the purpose of solving combinatory expensive problems, following a format similar to that of a developer diary. The main objective of this algorithm will be to find good solutions in a greatly reduced amount of time.

Initially, a mathematical frame necessary to understand how to solve unconstrained minimization problems and how and why to use statistical tests (and which ones to use) for the comparison of the different algorithms will be presented.

This will be followed by an exposition fo the classical genetic algorithms that have been used as a basis for the design of the algorithm to be developed and the modifications that will be introduced to it as the algorithm development progresses will be presented; both of them will contain asssociated pseudocode for an easier interpretation.

The design of the target algorithm will be carried out as follows. First, a genetic algorithm is going to be chosen as the base algorithm and, once its performance is analyzed, a series of modifications are going to be implemented, which, at the end, will compose the final version of the algorithm. These modifications are going to be justified by observing and analyzing the results obtained through the previous attempts with the objective of finding ways of making the most of the few iterations available in these kind of problems. Additional tasks regarding the development of this type of algorithms are described in detail. Furthermore, experimental analyses will be performed to show that successive version of the algorithm improve on the previous ones.

Since there are no specific algorithms for solving expensive combinatory problems, we are not able to compare the final results obtained with any kind of benchmark in order to obtain conclusions. Therefore, the analysis of the performance of the developed algorithm will be based on the comparison with the results of the previous developed versions and an analysis of the percentage of improvement regarding the classical genetic algorithm that has been used as a basis. The conclusions reached indicate that the algorithm presented within this project is highly performant.

Yo, **Irene Trigueros Lorca**, alumna de la titulación Doble Grado en Ingeniería Informática y Matemáticas de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada** y de la **Facultad de Ciencias**, con DNI 77385991F, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca de ambos centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Irene Trigueros Lorca

Granada, a 26 de junio de 2023.

D. **Daniel Molina Cabrera**, Profesor del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

D. **Francisco Herrera Triguero**, Profesor del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Diseño de Metaheurística para problemas combinatorios costosos***, ha sido realizado bajo su supervisión por **Irene Trigueros Lorca**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 26 de junio de 2023.

Los directores:

Daniel Molina Cabrera **Francisco Herrera Triguero**

Agradecimientos

En primer lugar, quiero agradecer y dedicar este proyecto a mis padres porque gracias a ellos he conseguido llegar a este punto de mi vida, por apoyarme en todo lo que pueden, por educarme para ser una persona que ponga todos sus esfuerzos en conseguir lo que se propone y por las incontables horas de llamadas a lo largo de estos cinco años aunque fuese solo para preguntarme si me iba bien o tenía algún problema.

También a mi hermana que siempre sabe como animarme y me ha inspirado todos estos años a mejorar como persona para poder ser la mejor influencia para ella.

A todos los amigos que he hecho en la universidad, por todas las horas de estudio, por todo el apoyo y la ayuda que nos hemos aportado a lo largo de los años, por todas las conversaciones absurdas solo para distraernos de los problemas... Empezaron como desconocidos y se convirtieron en una segunda familia.

Finalmente, agradecer a mis tutores, ya que sin ellos este trabajo no habría sido posible. Pero sobre todo agradecer a Daniel Molina Cabrera, por haberme guiado y aconsejado a lo largo del desarrollo de este trabajo, así como su tiempo dedicado para ello, por haberme ayudado siempre que lo necesitaba aunque no tuviese mucho tiempo.

Índice general

Índice de figuras

Capítulo 1: Introducción

Cuando se debe afrontar un problema, la variedad de aproximaciones a seguir resulta ser muy amplia y su elección depende en gran medida de dos factores fundamentales: qué clase de solución se desea extraer del problema, y de qué recursos se dispone para ello. Por ello, la rama más clásica de la computación siempre ha tratado de resolver los problemas presentados de forma exacta. Es decir, ha tratado cada problema como si solo existiera una sola solución al mismo, la óptima. Esta forma de pensamiento se basa en la confianza que se tiene en que los problemas que tradicionalmente eran irresolubles para los humanos, serían más accesibles para los computadores, gracias a su capacidad superior de cómputo pesado.

Esto último resulta cierto en muchos escenarios, fundamentalmente en lo referente a los problemas más puramente matemáticos: operaciones que un humano podría tardar años en resolver a mano, un ordenador podría resolverlas en cuestión de minutos. Sin embargo, la mayoría de problemas que nos encontramos en el mundo real son complejos y difíciles de resolver, lo que implica que no se pueda dar con la solución óptima en un tiempo razonable.

Uno de los problemas más conocidos capaz de ilustrar este hecho es el **Problema del Viajante de Comercio**. Este problema consiste en que, dado un conjunto de ciudades por las que el comercial debe pasar, se debe encontrar el orden en que visita las ciudades de forma que el comercial recorra la menor distancia posible. Aunque sea un problema sencillo de formular, la cantidad de posibles caminos incrementa en gran medida a la vez que el número de ciudades a recorrer aumenta. Por tanto, al aumentar el tamaño del problema, toda técnica conocida para extraer la solución exacta requeriría de un tiempo de ejecución que deja de ser asequible incluso para los ordenadores. A este tipo de problemas se les denominan problemas NP (*Nondeterministic Polynomial Time*), por lo que no pueden ser resueltos de forma realista exacta en tiempo polinomial.

Por ello, tenemos que encontrar alternativas más viables para la resolución de los problemas, es decir, tenemos que usar algoritmos aproximados, que proporcionan buenas soluciones (no necesariamente la óptima) en un tiempo razonable. Esto es, haremos uso de estrategias de diseño generales para procedimientos heurísticos de resolución de problemas: las metaheurísticas.

Los problemas de optimización costosa (EOP) o **problemas *expensives*** se refieren a los problemas que requieren costes elevados, o incluso inasequibles, con el fin de evaluar los candidatos a soluciones. Este tipo de problemas existen en una gran cantidad, y cada vez con más frecuencia, de aplicaciones significativas del mundo real. Un tipo de problemas *expensive* podrían ser los problemas de optimización de redes neuronales profundas utilizando metaheurísticas. Un ejemplo esto se podría encontrar en el artículo [1]; en ese estudio se proponen y se comparan tres métodos híbridos en combinación con el popular clasificador con redes neuronales para el modelado de incendios forestales.

También, cabe destacar que "coste elevado" es más un concepto relativo que uno absoluto en la mayoría de problemas del mundo real. Por ejemplo, en situaciones dinámicas, como podría ser

recalcular una ruta porque se haya cortado una calle o porque se ha producido un atasco, o incluso situaciones de emergencia como epidemias o desastres naturales, transporte y envío de materiales para operaciones diarias importantes para salvar vidas, etc., el coste de optimización en situaciones normales se convierte en un coste demasiado elevado. Para el caso de los problemas de parámetros reales nos encontramos con que se están planteando cada vez más algoritmos especialmente diseñados para problemas *expensive* (por ejemplo, que la función de evaluación dependa de una simulación).

Ahora bien, cada vez es más común usar problemas combinatorios en problemas complejos, lo que implica un mayor coste de evaluación. Un ejemplo de esto lo podemos encontrar en el artículo [2], donde en su introducción se detalla el por qué el *Well Placement Optimization Problem* (WPOP) en el desarrollo y gestión de campos petroleros se considera un problema *expensive* (se debe al cálculo tan complejo de ecuaciones diferenciales para predecir la influencia de la estrategia de producción en las propiedades geológicas y petrofísicas de la reserva). Si bien hemos comentado que en el caso de parámetros reales (caso continuo) se han propuesto algoritmos específicos, esto no ha sido el caso para el ámbito de los problemas combinatorios *expensive*, para los que no se han encontrado ninguna referencia. Por lo que es de gran interés crear un algoritmo que resulte útil para este tipo de situaciones, con el fin de reducir los costes todo lo posible.

En este Trabajo de Fin de Grado vamos a diseñar, implementar y proponer un algoritmo especialmente diseñado para problemas combinatorios *expensive*. El objetivo principal de este algoritmo será encontrar buenas soluciones en una cantidad de tiempo bastante reducida, lo que vamos a traducir en evaluar un número muy reducido de soluciones. Para ello, procederemos a tomar un problema sobre el que trabajar y extraer conclusiones y un algoritmo de referencia sobre el que realizaremos un proceso recursivo: introduciremos alguna modificación y compararemos los resultados obtenidos con los del algoritmo de referencia, en caso de mejorarlos, este algoritmo con modificación se convertirá en el nuevo algoritmo de referencia. De esta forma se puede garantizar que los resultados que finalmente alcanzaremos son competitivos.

1.1. Objetivos

Los objetivos principales se centrarán en el desarrollo de una metaheurística útil para aplicarse a problemas combinatorios costosos:

1. Parte matemática: Revisión teórica en la que se basa el desarrollo de algoritmos para problemas de minimización. Dentro del mismo, encontramos los siguientes objetivos parciales:
   - Repaso de distintos algoritmos de la literatura para resolver problemas de minimización sin restricciones.
   - Exposición de algunos resultados interesantes a considerar sobre convergencia.
   - Exposición de diversos *tests* estadísticos para comparar algoritmos.
2. Parte informática: Propio diseño experimental del algoritmo en cuestión. Dentro del mismo, encontramos los siguientes objetivos parciales:
   - Exposición de los algoritmos clásicos que se han usado como base.
   - Diseño de varias modificaciones y mejoras a introducir a los algoritmos base para hacerlos más competitivos en nuestro problema.

- Evaluación experimental de los distintos algoritmos ejecutándolos sobre distintos conjuntos de instancias del problema concreto con el que trabajaremos, analizando los resultados obtenidos y comparándolos entre sí.

1.2. Estructura de la memoria

Este trabajo se divide en dos partes, y estas a su vez en varios capítulos. Además, se incluyen tres capítulos al comienzo. Este ha sido el Capítulo 1, y a continuación encontraremos el Capítulo 2, dedicado a la planificación y presupuesto del proyecto; y el Capítulo 3, dedicado a dar un repaso bibliográfico de información necesaria para introducir el trabajo.

La Parte I, más vinculada con la parte matemática del trabajo, está dedicada a analizar la base teórica matemática sobre la que se construye el desarrollo de algoritmos cuyo objetivo es alcanzar una solución óptima. Tras la introducción del problema general de optimización a tratar y algunas definiciones importantes en el Capítulo 4, en el Capítulo 5 se presentan varios algoritmos que traten de resolver un problema de minimización sin restricciones. Además, en el Capítulo 6 se presentan y explican una serie de test estadístico, algunos de los cuales utilizaremos posteriormente para analizar si se producen mejoras significativas entre las distintas versiones del algoritmo que se desarrollará. Esta parte finaliza en el Capítulo 7, donde se desarrolla el problema concreto con el que se trabajará, aportando una definición matemática e información sobre los datos de dicho problema que se utilizarán.

La Parte II, más vinculada con la parte informática del trabajo, está dedicada a explicar y mostrar mediante pseudocódigo los distintos algoritmos y diversas modificaciones introducidas que se han usado durante el diseño de nuestra metaheurística junto con un análisis de los resultados obtenidos. En el Capítulo 8 se presentan los dos algoritmos de referencia que se han usado de base para el desarrollo de un algoritmo competitivo para problemas combinatorios *expensive*, mientras que en el Capítulo 9 se presentan los distintos componentes que se han ido desarrollando para aplicarlos como modificaciones sobre los algoritmos anteriores. Por último, en el Capítulo 10 se proporcionarán los distintos parámetros utilizados y se realizará un análisis completo del desarrollo de nuestro algoritmo justificando por qué cada modificación ha sido introducida.

Finalmente, en el Capítulo 12 se encuentran las conclusiones que se han obtenido tras realizar todo el trabajo. Adicionalmente, se incluirá la bibliografía y un apéndice donde se encontrarán todas las tablas de resultados de los distintos algoritmos que se han probado en la parte experimental de este proyecto.

Capítulo 2: Estimación y Presupuesto

En este capítulo se detalla cómo se ha organizado el trabajo y el tiempo dedicado a cada una de ellas. Además, se realiza una estimación del presupuesto necesario para desarrollar el proyecto.

2.1. Planificación

La planificación previa de este proyecto se ha realizado siguiendo una metodología ágil. Es el modelo de planificación que más se ajusta a este tipo de trabajo, ya que, *a priori*, se desconoce la dificultad de las tareas a realizar.

Sin embargo, es cierto que antes de empezar el trabajo se estableció una planificación base bastante amplia para poder asegurar que se iba a finalizar el proyecto a tiempo.

2.1.1. Planificación Base

La planificación inicial que se estableció antes de iniciar el proyecto se puede expresar mediante la información representada en la tabla 2.1:

Tabla 2.1: Planificación Base

Resumen	**Tareas**	**Planificación**
Investigar el problema	Obtener instancias del problema Buscar algoritmos que lo resuelvan Buscar posibles implementaciones	Noviembre
Algoritmos de referencia	Elegir un algoritmo como referencia y estudiarlo Reducir el problema y meter equilibrio	Enero-Febrero
Experimentación	Formas de inicialización no aleatorias → Diseño experimental	2 semanas
Modificaciones	Propuesta de modificaciones Obtención de resultados	Marzo-Abril
Memoria	Escribir el informe	Mayo

2.1.2. Planificación Final

Tareas realizadas

Si bien es cierto que se han realizado una gran cantidad de tareas (sobre todo distintas modificaciones sobre algoritmos), con el fin de mantener la simplicidad, se han agrupado algunas tareas que tenían funciones similares. Esta agrupación también es de ayuda para simplificar la estimación del tiempo que se le ha dedicado a cada una de las tareas. Así, las tareas realizadas se resumen en la siguiente lista:

1. **Planteamiento y comprensión del problema**: Revisión del trabajo a realizar y reuniones con los tutores para proponer modificaciones y comprender mejor y aclarar todos los matices del Trabajo de Fin de Grado.
2. **Búsqueda de información y lecturas**: Búsqueda y lectura comprensiva de todos los artículos y documentos necesarios para la realización del proyecto.
3. **Planificación del proyecto**: Planificación de algunos aspectos que usar como base, así como las tareas que eran necesarias inicialmente. También hace referencia a partes de reuniones con los tutores para modificar las planificaciones (añadiendo o eliminando tareas) dependiendo del progreso alcanzado y los resultados obtenidos.
4. **Implementación de la propuesta inicial**: Implementación del código de los algoritmos base.
5. **Adaptación de los algoritmos base**: Modificación del código de los algoritmos base para adaptarlos al problema en cuestión.
6. **Modificación de los algoritmos**: Sucesivas modificaciones sobre el algoritmo base y los algoritmos que mejores resultados proporcionaban con el fin de mejorarlos aún más.
7. **Obtención de resultados**: Ejecución del código para obtener todos los resultados y cambiarlos de formato para su posterior análisis.
8. **Análisis de los resultados**: Interpretación de los resultados obtenidos.
9. **Revisión de la parte experimental**: Una vez dada por finalizada la parte experimental, se ha hecho una revisión exhaustiva de los códigos y de los resultados obtenidos.
10. **Elaboración de la memoria**: Desarrollo del informe.
11. **Revisión de la memoria**: Una vez terminado el trabajo, se ha hecho una revisión exhaustiva de la memoria.

Téngase en cuenta hay tareas que se han realizado casi simultáneamente, como serían la “Modificación de los algoritmos”, “Obtención de los resultados” y “Análisis de los resultados”. Esto se debe a la necesidad de saber cómo han influido las modificaciones para empezar a estudiar qué otra modificación podría ser beneficiosa. Por ejemplo, si se converge rápidamente a una solución, hay que estudiar por qué ha pasado y, una vez hecha la hipótesis, estudiar qué se podría modificar para que no suceda.

Una estimación del tiempo (en horas) dedicado a cada tarea se puede encontrar en la tabla 2.2.

Tiempo dedicado

Tabla 2.2: Tiempo dedicado

Actividad	**Duración (horas)**
Planteamiento y comprensión del problema	20
Búsqueda de información y lecturas	15
Planificación del proyecto	5
Implementación de la propuesta inicial	10
Adaptación de los algoritmos bases	5
Modificación de los algoritmos	200
Obtención de resultados	50
Análisis de los resultados	10
Revisión de la parte experimental	5
Elaboración de la memoria	170
Revisión de la memoria	10
Total	500

2.2. Presupuesto

Si quisiéramos valorar económicamente el proyecto, tenemos que tener en cuenta dos aspectos fundamentales: el precio de la mano de obra y el de cómputo como si tuviésemos que pagarlo. Además, también se incluirá el precio del ordenador en el que se ha realizado todo el trabajo. En la tabla 2.3 se puede ver un resumen de los cálculos realizados junto con el presupuesto final necesario.

El precio de la mano de obra son 25€ la hora.

El ordenador portátil usado para la realización de las ejecuciones ha sido un Asus Tuf Gaming A15 FA506IU-HN278 con un procesador AMD® Ryzen™ 7 4800H APU y 16GB (8GB×2) de RAM. Tras una breve búsqueda se puede comprobar que el precio actual de dicho portátil sería de 1300€.

Para calcular el coste de tiempo de cómputo es ha utilizado la calculadora de precios de Amazon Web Services (AWS Pricing Calculator). Elegiremos el servicio EC2, y dentro de eso elegimos las características más similares al ordenador utilizado; lo que nos lleva a la instancia c5d.2xlarge. Comprobamos el coste por hora del uso de dicho servicio y tenemos que este sería 0.48$/hora, es decir, aproximadamente 0.443€/hora.

Teniendo todos estos datos, podemos proceder a calcular cuál sería el presupuesto final de del proyecto:

$$450h \cdot 25\frac{€}{h} + 1300€ + 50h \cdot 0.443\frac{€}{h} = 12572.15€$$

Tabla 2.3: Cálculo del presupuesto

Concepto	**Precio base (€)**	**Tiempo (h)**	**Precio total (€)**
Mano de obra	25	450	11250
Ordenador	1300	—	1300
Tiempo de cómputo	0.443	50	22.15
Total			**12572.15**

En resumen, el **presupuesto de este proyecto queda fijado en 12572.15€**.

Capítulo 3: Repaso Bibliográfico

En esta sección se lleva a cabo una revisión bibliográfica sobre el tema en el que hemos centrado el proyecto. Buscamos con ello, llevar a cabo un pequeño recordatorio para poder entender los distintos ámbitos que más se han tratado y centrado los estudios en cuanto al diseño de metaheurísticas, las cuales se usan para resolver problemas cada vez más complejos. Posteriormente utilizaremos estos conocimientos para nuestro propio diseño de una metaheurística útil para problemas combinatorios *expensive*.

3.1. Contexto Bibliográfico

La computación evolutiva (*evolutionary computation*, EC) es un área de la ciencia de la computación que usa ideas de la evolución biológica para resolver problemas computacionales. La evolución es, en efecto, un método de búsqueda entre un número enorme de posibilidades de "soluciones" que permitan a los organismos sobrevivir y reproducirse en sus ambientes. También se puede ver la evolución como un método de adaptación a un entorno cambiante. En [3] nos podemos encontrar con un resumen del desarrollo de la computación evolutiva en el tiempo junto con aplicaciones reales de algoritmos evolutivos (tanto comerciales como científicas), como podría ser el uso de programación genética para mejorar estrategias óptimas de recolección. Si consideramos la computación evolutiva como un medio para encontrar buenas soluciones (aunque no sean las óptimas) dado un problema de optimización, es natural considerar que su hibridación con métodos de optimización existentes resultará en mejorar su rendimiento al explotar sus ventajas. Tales métodos de optimización se refieren desde algoritmos exactos estudiados en programación matemática [4] hasta algoritmos heurísticos hechos a medida para unos problemas dados. Los llamados algoritmos metaheurísticos también tienen un objetivo similar, y pueden ser combinados con EC, incluso si ambos enfoques suelen competir entre si. En el libro [5] se presentan varias posibilidades de combinar ECs con métodos de optimización, poniendo énfasis en la optimización de problemas combinatorios, tales como métodos *greedy*, construcciones heurísticas de soluciones factibles, programación dinámica, etc.

Una de las versiones de algoritmos evolutivos más utilizadas son los algoritmos genéticos (AG), que será en el que nos centremos ya que supondrá ser un algoritmo base para el desarrollo de este proyecto. Los algoritmos genéticos son algoritmos basados en poblaciones que se pueden describir como la combinación de dos procesos: la generación de elementos del espacio de búsqueda (recombinación o mutación de la población) y la actualización (selección y redimensionamiento) para producir nuevas soluciones basadas en el conjunto de puntos que se crean junto con los de la anterior población [6] [7]. En [3] se presenta una lista de componentes para la versión más simple de un AG, que se puede resumir en:

- Una población de soluciones candidatas a un problema dado, cada una codificada de acuerdo

a un esquema de representación elegido.

- Una función *fitness* que asigna un valor numérico a cada cromosoma (solución) de una población para medir su calidad como candidato a solución del problema.
- Un conjunto de operadores que se aplicarán a los cromosomas para crear una nueva población. Estos suelen incluir:
  - Operador de Cruce: Dos cromosomas padres recombinan sus genes para producir una o más soluciones hijas.
  - Mutación: Uno o varios genes de una solución se modifican de forma aleatoria.

Una explicación más completa y detallada de lo relativo a AGs se puede encontrar en [8]. En dicho capítulo también presentan algunos artículos y libros donde encontrar aplicaciones de AGs exitosas para la optimización de problemas combinatorios, entre ellas se destaca [9], la cual lista algunas de las referencias más útiles y accesibles que podrían ser de interés para gente experimentando con metaheurísticas, como podrían ser el problema del viajante de comercio, problemas relacionados con grafos, problema de la mochila en forma binaria, etc.

Cualquier algoritmo que siga un ciclo reproducción-recombinación en una población de estructuras se puede denominar un AG en el sentido más amplio del término. Sin embargo, desde el trabajo de Holland (1975) para clasificar cualquier algoritmo como un AG en un sentido más estricto, se debía demostrar que su comportamiento de búsqueda reflejaba lo que Holland llamaba un "paralelismo implícito". Eshelman [10] comprueba que CHC, un algoritmo genético no tradicional, ciertamente muestra paralelismo implícito. Incluso justifica que algunas de las características que parecerían descalificarlo como un verdadero GA no solo no lo descalifican, sino que lo hacen más potente que un AG tradicional. Eshelman describe en detalle los componentes (prevención de incesto, cruce HUX, reinicios de población...) y las características de CHC, lo contrasta con un AG tradicional, aporta una justificación teórica de sus diferencias y provee algunas comparaciones empíricas. Un ejemplo donde se aplique el algoritmo CHC para resolver un problema lo podemos encontrar en el artículo [11], donde se estudia el uso de tres estrategias de memoria explícita combinadas con el algoritmo CHC para resolver distintas instancias del problema del viajante de comercio dinámico.

Llamaremos **problemas de optimización costosos o *expensive*** (EOP) a aquellos problemas que requieran costes elevados o incluso inalcanzables para evaluar candidatos a soluciones. Existen una gran cantidad de EOPs relativos a aplicaciones en el mundo real; además, con el progreso de la sociedad y problemas emergentes como las *smart cities*, la era del *big data*, etc., la resolución más eficiente de EOPs se ha vuelto cada vez más esencial para prosperar en distintos campos. Sin embargo, debido al coste tan elevado de cálculo, es complicado para los algoritmos de optimización encontrar una solución satisfactoria a dichos EOPs. Por ello, la EC se ha adoptado en gran medida para resolver EOPs, llevando a un campo de investigación de rápido crecimiento. Hasta la fecha, se han conducido varias investigaciones sobre el uso de EC para EOP y han alcanzado un éxito considerable [12] [13] [14] [15]. El concepto de EOP se suele mencionar junto con algunos conceptos del problema similares, como podría ser un problema de optimización a gran escala. En [12] se hacen referencia y se explican varios de estos conceptos.

Sin embargo, si bien somos capaces de encontrar una gran cantidad de investigaciones para EOPs, debemos notar que la vasta mayoría se refieren a problemas con parámetros reales, es decir, que pertenecen a los casos continuos. Esto implica que para el ámbito de los problemas combinatorios *expensive* nos encontramos con una sequía de estudios al respecto y, por tanto, algoritmos desarrollados para optimizar su resolución. Un ejemplo de este tipo de problemas lo podemos encontrar en el artículo [16], donde se emplea una fusión de algoritmos de optimización basados en

la biología y modelos de *Deep Learning*. Para recalcar la necesidad de algoritmos pensados para problemas combinatorios *expensive* podemos mencionar el artículo [17], donde se propone el uso de EvoPruneDeepTL (un modelo de poda evolutiva para aprendizaje por transferencia basado en redes neuronales profundas) aplicado a un problema de clasificación de imágenes. En dicho artículo los propios autores reconocen que estuvieron obligados a reducir en gran medida el número de evaluaciones de los algoritmos utilizados, lo que impedía el uso de tests estadísticos para estudiar el nivel de significación de las diferencias encontradas, debido a los elevados tiempos de ejecución para cada simulación.

Los AG son algoritmos de búsqueda potentes que pueden ser aplicados a un amplio rango de problemas. Generalmente, el establecimiento de los parámetros se realiza antes de ejecutar un AG y esta configuración se mantiene constante durante toda la ejecución. Sin embargo, es interesante considerar el uso parámetros auto-adaptativos. Es decir, tomar un enfoque en el que el control de los parámetros de un AG pueda ser codificado dentro de los cromosomas de cada individuo. Así, los valores son totalmente dependientes del mecanismo de evolución y del contexto del problema. Pellerin et al. [18] realizan un estudio sobre este tipo de comportamiento y obtienen resultados que indican un enfoque prometedor de desarrollo de AGs con parámetros auto-adaptativos que no requieran que el usuario los pre-ajuste. Podemos también encontrar una breve clasificación de los distintos métodos de auto-adaptabilidad propuestos en la literatura: empírico y adaptativo [19] [20]:

- Enfoque empírico: Se basan solo en la experimentación y en la observación. Miden el rendimiento de las AG por ensayo y error, a la vez que se modifican los parámetros del algoritmo [19]. Por lo tanto, es necesario encontrar buenos valores para dichos parámetros antes de ejecutar el algoritmo, y estos valores se mantendrán constantes durante la ejecución.
- Enfoque adaptativo. Nos encontramos con dos categorías:
  - Parámetros adaptativos limitados: Analiza los efectos aislados de uno o dos parámetros teniendo en cuenta el resto [21].
  - Parámetros auto-adaptativos: Refiriéndose a las técnicas que permite la evolución o adaptación de diversos parámetros de AG durante su ejecución [22] [23].

Como ya se ha indicado, los AGs tienen gran capacidad de adaptarse a la estructura del espacio de soluciones y controlar el equilibrio entre búsquedas locales y globales, incluso sin ajustar sus parámetros, como serían la probabilidad de cruce o mutación. Obviamente, ningún algoritmo es perfectamente adecuado para todos los problemas de acuerdo al teorema *No free lunch* [24], pero un determinado tipo de cruce puede llegar a resolver un gran número de problemas. Por ello, se han propuesto una gran cantidad de variantes de AGs en las que se modifican el tipo de cruce a lo largo de estos últimos años. Chaturvedi y Sharma en el capítulo *A Modified Genetic Algorithm Based on Improved Crossover Array Approach* [25] estudiaron la introducción de un AG con un operador de cruce mejorado de tal forma que se eligiese dinámicamente el tipo de operador de cruce que se iba a utilizar en el problema. Cheng y Gen [26] propusieron un nuevo operador de cruce para el problema del viajante de comercio con el objetivo de mejorar el AG tanto en tiempo como en precisión mediante el uso de relaciones de precedencia local entre los genes para obtener candidatos de posibles *edges* y relaciones de precedencia global entre los genes para obtener al mejor entre los candidatos. Quiobing y Lixin [27] propusieron un nuevo mecanismo de cruce para AG con cromosomas de longitud variable para problemas de optimización de caminos.

Los métodos de selección de padres son básicos dentro de los AGs, ya que determinan las posibilidades de los individuos de influir en las generaciones siguientes. Dado que en los AGs el

operador que produce diversidad es el cruce, es necesaria una adecuada selección de los individuos para poder conseguir un adecuado equilibrio entre exploración y explotación. En este trabajo se utilizarán dos métodos:

- Una Selección por Torneo clásica, donde se eligen de forma aleatoria cierta cantidad de individuos de la población y solo elegimos el mejor.
- Emparejamiento Variado inverso (*Negative Assortative Mating*, NAM) [28]. Está orientado a generar diversidad, eligiéndose un elemento de forma aleatoria y otro grupo de individuos; de este grupo elegiremos como segundo padre a aquel elemento más diferente del primero.

En los AG estacionarios se debe determinar los individuos que serán reemplazados por los nuevos individuos generados. Esta elección puede determinar el nivel de presión selectiva que se exige a los individuos para ser introducidos o permanecer en la población, y la pérdida de diversidad que se introduce conforme el proceso de búsqueda avanza. Se han propuesto numerosas estrategias de reemplazo, algunas como introducir alta presión selectiva, como sería el reemplazar el peor elemento de la población [29], otras para mantener una alta diversidad, como serían los métodos de multitud o *crowding* [30] que buscan que las nuevas soluciones reemplacen las soluciones parecidas a ellas, o un equilibrio entre ambas.

El problema de la mochila cuadrático (QKP, *quadratic knapsack problem*) binario fue introducido por Gallo et al. [31] y resulta una generalización del clásico problema de la mochila (KP, *knapsack problem*), donde el beneficio que aporta cada elemento deja de ser solo individual, pudiendo aportar más valor si se combina con otro elemento. Aunque el QKP no se ha estudiado de forma tan intensiva como el problema de la asignación cuadrática, nos podemos encontrar con numerosos artículos a lo largo de estos últimos años relativos a este problema. Como cabría esperar debido a su generalidad, el QKP tiene un rango de aplicaciones muy amplio. Witzgall [32] presentó un problema que surge en telecomunicaciones cuando se debe seleccionar un número de localizaciones para las estaciones de satélites de forma que el tráfico global entre estas estaciones se maximice y la restricción de presupuesto se respete. Este problema resulta ser un QKP. Nos encontramos con modelos similares al considerar la localización de aeropuertos, estaciones de tren o terminales de manejo de carga [33]. Ferreira et al. [34] consideran un problema de diseño VLSI (*Very Large Scale Integration*) donde los grafos de gran tamaño necesitan ser descompuestos en grafos de más pequeños de tamaño manejable. El problema de optimización asociado para un único subgrafo puede ser reconocido como un QKP de minimización. Pisinger [35] realizó un estudio sobre los límites superiores presentados en la literatura y muestra la estanqueidad relativa de los diversos límites, proporcionando resultados y un estudio experimental donde se comparan estos resutalos con respecto a la potencia y el esfuerzo computacional.

Por último, se debe mencionar que se ha intentado seguir una guía de buenas prácticas (artículo [36]), aunque hay un punto que no se ha podido cumplir: el uso de benchmarks, debido a la ausencia de algoritmos diseñados específicamente para resolver este tipo de problemas. Podemos resumir lo utilizado de dicha guía en:

- **Validación de los resultados**: En tanto que no existen benchmarks que podamos utilizar para este trabajo, no podemos comparar nuestros resultados con otros algoritmos. Sin embargo, se realizarán tests estadísticos no paramétricos ([37]) entre las diferentes versiones del algoritmo que vamos a desarrollar para poder asegurar que realmente se están produciendo mejoras.
- **Análisis de los componentes y ajuste de parámetros de la propuesta**: Este análisis y justificación se realizará a lo largo de este trabajo.

- **Utilidad del algoritmo propuesto**: Como ya se ha indicado anteriormente y se volverá a indicar más tarde, no hay algoritmos que estén orientados a resolver problemas de optimización combinatorios *expensive*, por lo que este proyecto no es solo útil, sino que también supone una novedad.

Parte I

Parte matemática

Capítulo 4: Introducción

Los algoritmos usan operadores matemáticos con el objetivo de alcanzar una solución a los problemas a los que son aplicados y también es necesario obtener el óptimo. La prueba de este hecho puede ser realizada gracias a la teoría de optimización del Análisis Numérico. Nos centraremos en esta teoría, aportando resultados que nos permitan saber si un punto es el óptimo de una función y su relación con los algoritmos básicos de optimización global.

La programación no lineal es un área de las matemáticas aplicadas que involucra problemas de optimización cuando las funciones son no lineales. Nuestro objetivo es introducir este problema y revisar las condiciones generales de optimalidad, que son la base de muchos algoritmos por sus soluciones. Tras esto, aportaremos numerosas nociones relativas al rendimiento de los algoritmos en términos de convergencia, orden de convergencia y comportamiento numérico.

Cabe mencionar que aunque el problema abordado en este trabajo implica la maximización del valor de la solución, se utilizará la notación convencional y usual para tratar este tipo de problemas, es decir, la minimización del valor de la solución. Esto se debe a que, al fin y al cabo, son problemas equivalentes. Una forma sencilla de transformar un problema de maximización en uno de minimización es cambiar el símbolo de los valores. Por ello, en tanto que ambos problemas son equivalentes, se mantendrá la notación de minimizar la función objetivo.

4.1. Definición del problema

En primer lugar, daremos una definición a nuestro problema.

Definición 4.1 *Consideramos el problema de calcular el valor de un vector de variables de decisión $x \in \mathbb{R}^n$ que minimiza la función objetivo $f : \mathbb{R}^n \to \mathbb{R}$ donde x pertenece a un conjunto factible de soluciones $\mathcal{F} \in \mathbb{R}^n$. Consideramos el siguiente problema:*

$$\min_{x \in \mathcal{F}} f(x) \tag{4.1}$$

Nota: Llamaremos **conjunto factible** al espacio de soluciones, es decir, al conjunto de todos los puntos posibles de un problema de optimización que satisface las restricciones del problema.

Ahora, presentamos dos casos de ello:

- El conjunto factible de soluciones $\mathcal{F}$ es todo el espacio $\mathbb{R}^n$. En este caso, el problema es el siguiente:

  $$\min_{x \in \mathbb{R}^n} f(x) \tag{4.2}$$

  Diremos que el problema 4.1 es no restringido. De forma general, el problema 4.1 es no restringido si $\mathcal{F}$ es un conjunto abierto.

- El conjunto factible de soluciones está descrito por restricciones de desigualdad y/o igualdad en las variables de decisión:

$$\mathcal{F} = \{x \in \mathbb{R}^n : g_i(x) \leq 0, i = 1, ..., p; h_j(x) = 0, j = 1, ..., m\} \tag{4.3}$$

Entonces, el problema 4.1 se convierte en:

$$\begin{array}{c} min_{x \in \mathbb{R}^n} f(x) \\ g(x) \leq 0 \\ h(x) = 0 \end{array} \tag{4.4}$$

donde $g : \mathbb{R}^n \to \mathbb{R}^p$ y $h : \mathbb{R}^n \to \mathbb{R}^m$. En este caso, diremos que el problema es restringido.

El problema 4.1 es no lineal cuando al menos una de las funciones del problema es no lineal, es decir, f,g_i, $i = 1, ..., p$, h_j, $j = 1, ..., m$ es no lineal en x.

Normalmente, asumimos que en un problema del tipo 4.1 el número de condiciones de igualdad, m, es menor que el número de variables, n; en otro caso, el conjunto factible de soluciones será el vacío, a no ser que haya dependencia en las restricciones. Si solo hay condiciones de igualdad, el problema se llamará "**problema no lineal con restricciones de igualdad**". Equivalentemente, se tiene el caso de que solo aparezcan condiciones de desigualdad.

En lo siguiente asumiremos que nuestras funciones f,g,h son diferenciables y continuas en $\mathbb{R}^n$. Además, cuando f sea una función convexa y el conjunto $\mathcal{F}$ sea también convexo, al problema se le llamará "**problema convexo no lineal**". Particularmente, $\mathcal{F}$ es convexo si las funciones que nos aportan las restricciones de desigualdad son convexas y las funciones de las restricciones de igualdad son afines.

La convexidad nos permite añadir estructura a estos problemas y poder explotarlo desde un punto de vista teórico y computacional, esto se debe a que si f es una función convexa cuadrática y g,h son afines, entonces tenemos que tratar con un problema de programación cuadrática. Sin embargo, nos centraremos en problemas no lineales generales, sin asumir convexidad.

Definición 4.2 *Un punto un $x^* \in \mathcal{F}$ es un **solución global** del problema 4.1 si $f(x^*) \leq f(x)$, $\forall x \in \mathcal{F}$. El punto es una **solución global estricta** si $f(x^*) < f(x)$, $\forall x \in \mathcal{F}$, $x \neq x^*$.*

La existencia de soluciones globales se debe a la compacidad de $\mathcal{F}$, en relación con el teorema de Weierstrass:

Teorema 4.1 (Teorema de Weierstrass) *Sea $a, b \in \mathbb{R}$ con $a < b$ y sea $f : [a, b] \to \mathbb{R}$ una función continua. Entonces, el intervalo $f([a, b])$ es cerrado y acotado.*

Una consecuencia directa para los problemas sin restricciones es que una solución global existe si el conjunto $\mathcal{L}^\alpha = \{x \in \mathbb{R}^n \leq \alpha\}$ es compacto para un α finito.

Definición 4.3 *Un punto $x^* \in \mathcal{F}$ es una **solución local** del problema del problema 4.1 si existe x^* en un vecindario abierto $\mathcal{B}_{x^*}$ de x^* tal que $f(x^*) \leq f(x)$, $\forall x \in \mathcal{F} \bigcap \mathcal{B}_{x^*}$. Además, es una **solución local estricta** si $f(x^*) < f(x)$, $\forall x \in \mathcal{F} \bigcap \mathcal{B}_{x^*}$, $x \neq x^*$.*

Determinar una solución global a un problema de este tipo es normalmente una tarea complicada. Los algoritmos que resuelven este tipo de problemas se usan para alcanzar óptimos locales. Incluso en aplicaciones prácticas obtener este tipo de soluciones puede ser también algo bueno.

Ahora introduciremos **notación**:

- Dado un vector $y \in \mathbb{R}^n$, definimos su **traspuesta** por y'. Esta definición puede ser extendida a las matrices $A \in \mathbb{R}^n \times \mathbb{R}^n$.
- Dada una función $h : \mathbb{R}^n \to \mathbb{R}$, denotamos el vector **gradiente** por $\nabla h(x) = \left(\frac{\partial f(x)}{\partial x_1}, ..., \frac{\partial f(x)}{\partial x_n}\right)$ y la matriz **Hessiana** se denota por

$$\nabla^2 h(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{pmatrix}$$

- Dada una función vector $w : \mathbb{R}^n \to \mathbb{R}^q$, denotamos la matriz de $n \times q$ cuyas columnas son $\nabla w_j(x)$, $j = 1, ..., q$ con $\nabla w(x)$.
- Sea y un vector, $y \in \mathbb{R}^q$, denotamos la **norma Euclídea** por $||y||$. Supongamos que sus componentes son y_i, $i = 1, ..., q$ y sea A una matriz cuyas columnas sean a_j, $j = 1, ..., q$. Sea $\mathcal{K} \subset \{1, ..., q\}$ un subconjunto de índices. Denotamos por $y_\mathcal{K}$ al subvector de y con componentes y_i tales que $i \in \mathcal{K}$ y por $A_\mathcal{K}$ a la submatriz de A compuesta por las columnas a_j con $j \in \mathcal{K}$.

4.2. Condición de Optimalidad

Nuestras soluciones locales deben satisfacer algunas condiciones de optimalidad necesarias. Si nos referimos a problemas como 4.2, tenemos uno de los resultados más conocidos del Análisis Numérico clásico:

Proposición 4.1 *Sea x^* una solución local al problema 4.2, entonces*

$$\nabla f(x^*) = 0 \tag{4.5}$$

Además, si f es continuamente 2-diferenciable, entonces

$$y' \nabla^2 f(x^*) y \geq 0, \forall y \in \mathbb{R}^n \tag{4.6}$$

Nota: Diremos que una función es continuamente i-diferenciable si es diferenciable i veces y dichas diferenciales son continuas.

Si tenemos problemas como 4.1, la mayoría de las condiciones necesarias de optimalidad usadas en el desarrollo de los algoritmos asumen que una solución local debe satisfacer algunas de estas condiciones para evitar alcanzar casos degenerados. Estas condiciones se suelen llamar "**calificaciones de restricción**" y, entre ellas, la más simple y más comúnmente usada es el requisito de restricción de independencia lineal:

Definición 4.4 *Sea $\hat{x} \in \mathcal{F}$. Decimos que la restricción de desigualdad g_i está **activa** en $\hat{x}$ si $g_i(\hat{x}) = 0$. Denotamos por $\mathcal{F}_a(\hat{x})$ al conjunto de índices de las restricciones de desigualdad activas en $\hat{x}$:*

$$\mathcal{F}_a(\hat{x}) = \{i \in \{1, ..., p\} : g_i(\hat{x}) = 0\} \tag{4.7}$$

Nótese que en la definición anterior podemos concluir que las restricciones de igualdad h_j son activas en $\hat{x}$. La restricción de independencia lineal se satisface si el gradiente de las restricciones activas son linealmente independientes.

Bajo las condiciones de independencia lineal asumidas, los problemas de restricciones como 4.1 pueden resolverse usando la función Lagrangiana generalizada:

$$L(x, \lambda, \mu) = f(x) + \lambda' g(x) + \mu' h(x) \tag{4.8}$$

donde $\lambda \in \mathbb{R}^p$, $\mu \in \mathbb{R}^m$ son multiplicadores de Lagrange. El siguiente resultado nos dan información sobre la existencia de estos multiplicadores.

Proposición 4.2 *Sea x^* una solución local del problema 4.1 y supongamos que la independencia lineal se satisface en x^*. Entonces, los multiplicadores de Lagrange $\lambda^* \geq 0$, μ^* existe de tal forma que:*

$$\nabla_x L(x^*, \lambda^*, \mu^*) = 0 \tag{4.9}$$

y

$$\lambda'^* g(x^*) = 0$$

Además, si f, g, h son continuamente 2-diferenciables, entonces:

$$y' \nabla_x^2 L(x^*, \lambda^*, \mu^*) y \geq 0, \forall y \in \mathcal{N}(x^*)$$

donde

$$\mathcal{N}(x^*) = \{y \in \mathbb{R}^n : \nabla g'_{\mathcal{F}_a} y = 0; \nabla h(x^*)' y = 0\}$$

Si $x \in \mathcal{F}$ satisface una condición de optimalidad suficiente, entonces dicho punto es una solución local al problema 4.1. Para problemas generales, las condiciones de optimalidad suficientes pueden ser establecidas bajo la asunción de que las funciones problemas son continuamente 2-diferenciables, así que tenemos condiciones de suficiencia de segundo orden. Estas condiciones cambian si el problema a tratar es 4.2 o 4.1. Por lo tanto, los siguientes resultados mostrarán cuándo x^* son óptimos locales en ambos casos.

Proposición 4.3 *Asumimos que x^* satisface la condición 4.5. También asumiremos que*

$$y' \nabla^2 f(x^*) y > 0, \forall y \in \mathbb{R}^n, y \neq 0$$

es decir, se asume que $\nabla^2 f(x^*)$ es definida positiva. Entonces, x^* es una solución local estricta del problema 4.2.

Proposición 4.4 *Asumimos que $x^* \in \mathcal{F}$ y λ^*, μ^* satisfacen las condiciones de 4.9. Asumimos también que*

$$y' \nabla_x^2 L(x^*, \lambda^*, \mu^*) y > 0, \forall y \in \mathcal{P}(x^*), y \neq 0 \tag{4.10}$$

donde

$$\mathcal{P}(x^*) = \{y \in \mathbb{R}^n : \nabla g'_{\mathcal{F}_a} y \leq 0, \nabla h(x^*)' y = 0; \nabla g_i(x^*)' y = 0, i \in \mathcal{F}_a(x^*), \lambda_i^*\}$$

Entonces, x^* es una solución local estricta del problema 4.1.

Nótese también que $\mathcal{N}(x^*) \subseteq \mathcal{P}(x^*)$ y la igualdad se da si $\lambda_i^* > 0, \forall i \in \mathcal{F}_a(x^*)$.

Una característica importante y principal de los problemas convexos es que una solución global (estricta) del problema es también una solución local (estricta). Además, cuando f es (estrictamente) convexa, las funciones g_i son también convexas y las h_j son afines, entonces las condiciones de optimalidad necesarias dadas en términos de primeras derivadas parciales son suficientes para que un punto x^* sea una solución global (estricta).

Las condiciones de optimalidad son esenciales para problemas no lineales. Si conocemos la existencia de un óptimo global, entonces el método más común de obtenerlo es el siguiente:

1. Encontrar todos los puntos que satisfacen las condiciones necesarias de primer orden.
2. Tomas el óptimo global como el punto con el valor más bajo dado por la función objetivo
3. Si la función problema es 2-diferenciable, entonces comprueba la condición necesaria de segundo orden y elimina los puntos que no la satisfagan.
4. Para el resto de puntos comprobaremos la condición de suficiencia de segundo orden para encontrar el mínimo local.

Tenemos que destacar que este método no funciona en casos prácticos excepto por casos simples, esto se debe a que tenemos que calcular la solución de un sistema de ecuaciones dado por $\nabla f(x) = 0$ y este sistema es normalmente no trivial. Entonces, ¿dónde son importantes estas condiciones? Las condiciones de optimalidad son importantes en el desarrollo y análisis de algoritmos.

Un algoritmo que intenta resolver un problema dado por 4.1 genera una secuencia de soluciones factibles $x^k, k = 0, 1, ...$ y normalmente finaliza cuando se satisface un criterio de parada. Este criterio se suele basar en la satisfacción de condiciones de optimalidad necesarias dentro de una tolerancia prefijada. Adicionalmente, estas condiciones normalmente sugieren cómo mejorar la solución actual, con lo que la siguiente debería encontrarse más cercana al óptimo.

Así, las condiciones de optimalidad necesarias nos dan la base para el análisis de convergencia de algoritmos. Por lo tanto, las condiciones de suficiencia juegan un papel importante en el análisis del orden de convergencia.

4.3. Rendimiento de los algoritmos

4.3.1. Convergencia y Orden de Convergencia

Definición 4.5 *Sea $\Omega \subset \mathcal{F}$ el subconjunto de puntos que satisfacen las condiciones necesarias de optimalidad del problema 4.1. Un algoritmo se detiene cuando un punto $x^* \in \Omega$ es calculado. Por lo tanto, a este conjunto se le llamará* ***conjunto objetivo****.*

Un ejemplo de este conjunto para el problema sin restricciones podría ser $\Omega = \{x \in \mathbb{R}^n : \nabla f(x) = 0\}$; mientras que para el problema restringido este conjunto será el conjunto de puntos que satisfacen la condición 4.9.

Definición 4.6 *Sea $x^k, k = 0, 1, ...$ la secuencia de puntos generados por un algoritmo. Entonces, el algoritmo es* ***globalmente convergente*** *si un punto límite x^* de x^k existe tal que $x^k \in \Omega$ para cualquier punto de inicio $x^0 \in \mathbb{R}^n$.*

Diremos que el algoritmo es ***localmente convergente*** *si la existencia de* $x^* \in \Omega$ *solo puede ser establecida si el punto inicial,* x^0*, pertenece a un vecindario de* Ω*.*

La definición de convergencia establecida anteriormente es la más débil que asegura que un punto x^k arbitrariamente cercano a Ω puede ser obtenido con un k suficientemente grande.

En el caso de no tener restricciones, esto implica

$$\lim_{k \to \infty} ||\nabla f(x^k)|| = 0$$

El requerimiento más fuerte de convergencia nos muestra que la secuencia x^k converge a un punto $x^* \in \Omega$.

Ahora mostraremos cómo se puede definir el orden de convergencia de un algoritmo. Así, podemos asumir por simplicidad que los algoritmos generan una secuencia x^k que converge a un punto $x^* \in \Omega$. El concepto más empleado en términos de convergencia es el Q-orden de convergencia, el cual considera el cociente entre dos iteraciones sucesivas dado por

$$\frac{||x^{k+1} - x^*||}{||x^k - x^*||}$$

Entonces, podemos definir el siguiente tipo u orden de convergencia.

Definición 4.7 *El orden de convergencia es Q-lineal si existe una constante* $r \in (0, 1)$ *tal que*

$$\frac{||x^{k+1} - x^*||}{||x^k - x^*||} \leq r,$$

para cualquier k *suficientemente grande.*

Definición 4.8 *El orden de convergencia es Q-superlineal si*

$$\lim_{k \to \infty} \frac{||x^{k+1} - x^*||}{||x^k - x^*||} = 0$$

Definición 4.9 *El orden de convergencia es Q-cuadrático si*

$$\frac{||x^{k+1} - x^*||}{||x^k - x^*||^2} \leq R$$

para cualquier k *suficientemente grande y donde* $R > 0$ *es una constante.*

4.3.2. Comportamiento numérico

A pesar del rendimiento teórico que puedan tener los algoritmos, otro aspecto importante es el comportamiento práctico. En efecto, si tenemos una gran cantidad de operaciones algebraicas por operación, es posible superar una tasa de convergencia rápida. Tenemos numerosas medidas para evaluar el comportamiento numérico. Sin embargo, si la carga computacional (operaciones algebraicas por iteración) no es despreciable, entonces podemos usar algunas medidas dadas por el número de iteraciones, número de evaluaciones de funciones objetivo, etc.

Medir el rendimiento de los algoritmos es importante para problemas no lineales de gran escala. El término "gran escala" depende de la máquina que se encargue de los datos, pero ese tipo de problema son normalmente problemas sin restricciones que verifican que $n \geq 1000$, donde n es el número de variables. Sin embargo, un problema con restricciones se considerará ser un problema de gran escala cuando el número de variables es $n \geq 100$ y cuando la suma de las condiciones es 100 o mayor. Uno de los problemas más importantes es el trasladar algoritmos eficientes en problemas a pequeña escala a problemas de gran escala.

Capítulo 5: Programación no lineal sin restricciones

En este capítulo consideraremos algoritmos que traten de resolver el siguiente problema sin restricciones

$$\min_{x \in \mathbb{R}^n} f(x) \tag{5.1}$$

donde $x \in \mathbb{R}^n$ es el vector de variables de decisión y $f : \mathbb{R}^n \to \mathbb{R}$ es la función objetivo. Es lógico pensar que si somos capaces de resolver este problema, el procedimiento para ello puede ser ajustado a problemas con restricciones porque solo necesitaremos un conjunto abierto factible de soluciones $\mathcal{F}$, y un punto inicial $x^0 \in \mathcal{F}$.

Asumimos por simplicidad que nuestra función f es continuamente 2-diferenciable en $\mathbb{R}^n$. Además, asumiremos lo siguiente para asegurarnos de la existencia de una solución del problema 5.1: $\mathcal{L}^0 = \{x \in \mathbb{R} : f(x) \leq f(x^0)\}$ es compacto para algún $x^0 \in \mathbb{R}^n$.

5.1. Algoritmos de Optimización sin restricciones

Ahora presentaremos algunos modelos de algoritmos que serán usados para resolver los problemas previos. Estos tipos de algoritmos están caracterizados por generar una secuencia de puntos, $\{x^k\}$, empezando por un punto inicial x^0, usando la siguiente iteración

$$x^{k+1} = x^k + \alpha^k d^k \tag{5.2}$$

donde d^k es la dirección de búsqueda y α^k es el tamaño de paso junto con d^k. En este método tenemos dos parámetros a modificar: la dirección de búsqueda y el tamaño del paso, por lo tanto, dependiendo de como variamos estos datos, obtendremos diferentes métodos y esto afectará a las propiedades de convergencia. El tamaño de paso afecta a la convergencia global, mientras que la dirección de búsqueda afecta a la convergencia local.

El siguiente resultado nos da una relación entre convergencia y dirección de búsqueda:

Proposición 5.1 *Sea $\{x^k\}$ la secuencia generada por 5.2. También asumimos:*

1. *$d^k \neq 0$ si $\nabla f(x^k) \neq 0$.*
2. *$\forall k$ tenemos $f(x^{k+1}) \leq f(x^k)$.*
3. $$\lim_{k \to \infty} \frac{\nabla f(x^k)' d^k}{||d^k||} = 0 \tag{5.3}$$

4. $\forall k$ *con* $d^k \neq 0$, *tenemos* $\dfrac{|\nabla f(x^k)' d^k|}{||d^k||} \geq c||\nabla f(x^k)||$ *con* $c > 0$.

Entonces, tenemos que, o bien, existe $\hat{k}$ *tal que* $x^{\hat{k}} \in \mathcal{L}^0$ *y* $\nabla f(x^{\hat{k}}) = 0$, *o bien, se genera una secuencia infinita tal que:*

1. $\{x^k \in \mathcal{L}^0\}$.
2. $\{f(x^k)\}$ *converge.*
3. $$\lim_{k \to \infty} ||\nabla f(x^k))|| = 0 \tag{5.4}$$

La tercera condición implica que solo necesitamos una subsecuencia que tenga un punto límite en Ω. Este resultado nos da información sobre cómo es la convergencia en términos de la dirección de búsqueda.

Procedemos a describir dos métodos conocidos como algoritmos de Búsqueda Lineal y métodos basados en el gradiente.

5.2. Algoritmos de Búsqueda Lineal

Estos algoritmos están caracterizados por determinar el tamaño de paso α^k junto con la dirección de búsqueda d^k. El objetivo es elegir un tamaño de paso que asegure la convergencia de 5.2. Una elección puede ser elegir dicho tamaño de paso tal y como se describe en la siguiente ecuación:

$$\alpha^k = arg\min_{\alpha} f(x^k + \alpha d^k)$$

La ecuación anterior puede ser resumida en la siguiente idea: el tamaño de paso es el valor que minimiza la función objetivo junto con una dirección dada. Sin embargo, en esta situación es posible que el mínimo no pueda ser alcanzado porque la función necesita tener propiedades que nos permitan calcular dicho mínimo, por lo tanto, una búsqueda lineal no tiene por qué suponer la mejor solución computacionalmente hablando.

Por ello, tenemos que usar métodos de aproximación. Uno de ellos es calcular el gradiente de la función. En estos casos podemos asumir que

$$\nabla f(x^k)' d^k < 0, \forall k \tag{5.5}$$

Los algoritmos de búsqueda lineal más simples están proporcionados por Armijo, cuyo pseudocódigo se puede encontrar en Algoritmo 1.

La elección inicial de ∇^k se debe a la dirección d^k porque garantizamos que, en un número finito de pasos, α^k tiene un valor tal que $f(x^{k+1}) < f(x^k)$, por lo que las condiciones de convergencia de la proposición se encuentran satisfechas.

Esta búsqueda lineal encuentra un tamaño de paso que satisface la condición de decrecimiento suficiente de la función objetivo y, consecuentemente, el desplazamiento suficiente de la secuencia actual.

Algorithm 1 Algoritmo de Búsqueda Lineal de Armijo

procedure (Busqueda_Lineal)($\delta \in (0,1), \gamma \in (0,1/2), c \in (0,1)$)
  Elegir ∇^k tal que
$$\nabla^k \geq c\frac{|\nabla f(x^k)' d^k|}{||d^k||^2}$$
  $\alpha = \nabla^k$
  $N \leftarrow n$
  if $f(x^k + \alpha d^k) \leq f(x^k) + \gamma\alpha\nabla f(x^k)' d^k$ **then**
    $\alpha^k = \alpha$ y parar
  else
    Establecer $\alpha = \delta\alpha$ y volver al paso anterior.
  end if
end procedure

En algunos casos, necesitamos considerar que los algoritmos de búsqueda lineal no necesitan información sobre las derivadas. En estos casos, la condición 5.5 no puede ser verificada. Por lo tanto, la dirección d^k no tiene por qué ser descendiente.

Otra posible modificación al algoritmo propuesto podría ser sustituir la condición de parada por la siguiente, que no tiene información sobre la derivada:

$$f(x^k + \alpha d^k) \leq f(x^k) - \gamma\alpha^2||d^k||^2 \tag{5.6}$$

5.3. Métodos de Gradiente

El método de gradiente o método del descenso más rápido (*steepest-descent*) se considera un método básico de entre todos los algoritmos de optimización no restringidos. Tiene la regla básica de establecer $d^k = -\nabla f(x^k)$ en 5.2. Solo necesita información sobre la primera derivada y es importante debido a que el coste computacional y el almacenamiento disponibles son limitados.

Este método es un ejemplo de convergencia global, porque si usamos un algoritmo de búsqueda local adecuado, podemos establecer un resultado de convergencia global. El problema es su orden de convergencia, el cual es bajo y esto es la razón por la que este algoritmo no se suele usar solo. El esquema de este algoritmo es el siguiente:

Algorithm 2 Método de Gradiente

procedure (Gradiente)
  Establecer $x^0 \in \mathbb{R}^n$ y $k = 0$
  while $\nabla f(x^k)! = 0$ **do**
    Establecer $d^k = -\nabla f(x^k)$ y encuentra un tamaño de paso α^k usando 1
    Establecer $x^{k+1} = x^k - \alpha^k \nabla f(x^k)$ y $k = k + 1$.
  end while
end procedure

La elección inicial del tamaño de paso como la dada por Armijo es importante, ya que puede afectar al comportamiento del algoritmo. En términos de convergencia, el siguiente resultado nos asegura la convergencia sin necesitar asumir la compacidad del conjunto $\mathcal{L}^0$.

Nota: Sea $f : \mathbb{R}^n \to \mathbb{R}^m$. f se dice **Lipschitziana** si existe $K > 0$ tal que

$$||f(x) - f(y)|| \leq K||z - y||, \forall x, y \in \mathbb{R}^n$$

Proposición 5.2 *Si ∇f es Lipschitziana, continua y f está acotada inferiormente , entonces la secuencia generada por 2 satisface la tesis de 4.5.*

Como conclusión, con este resultados podemos comprobar que el gradiente es bueno en términos de convergencia global. Sin embargo, solo podemos probar la convergencia lineal. Desde un punto de vista práctico, el orden de convergencia es muy pobre y depende del número de condiciones de la matriz Hessiana.

5.4. Métodos de Gradiente Conjugado

Este algoritmo es muy popular debido a su simplicidad y sus bajos requerimientos computacionales. En efecto, solo necesita saber sobre las derivadas de primer orden. La idea principal es que la minimización de funciones cuadráticas estrictamente convexas en $\mathbb{R}^n$ como la siguiente

$$f(x) = \frac{1}{2}x'Qx + \alpha'x \tag{5.7}$$

donde Q es una matriz simétrica definida positiva, puede ser dividida en n minimizaciones sobre $\mathbb{R}$. Esto se puede hacer utilizando n direcciones, $d^0, ..., d^{n-1}$ conjugadas con respecto de la matriz Hessiana Q. Junto con cada dirección, se realiza una búsqueda lineal.

El siguiente algoritmo (Algoritmo 3) muestra todo lo mencionado anteriormente.

Se debe notar que el valor de α^k que se utilizará en el tercer paso es el que minimiza la función $f(x^k + \alpha d^k)$. Además, el cociente está bien definido porque para dos direcciones distintas tenemos $(d^j)Qd^i = 0$, con i, j tales que $i \neq j$.

Algorithm 3 Algoritmo de direcciones conjugadas para funciones cuadráticas

1: **procedure** (Direcciones_Conjugadas) (direcciones Q-conjugadas $d^0, ..., d^{n-1}$)
2: Establecer $x^0 \in \mathbb{R}^n$ y $k = 0$
3: **while** $\nabla f(x^k)! = 0$ **do**
4: Establecer $\alpha^k = \dfrac{\nabla f(x^k)'d^k}{(d^k)'Qd^k}$
5: Establecer $x^{k+1} = x^k - \alpha^k \nabla f(x^k)$ y $k = k + 1$.
6: **end while**
7: **end procedure**

En este algoritmo, las direcciones provienen de los datos, mientras que en el algoritmo del gradiente conjugado se calculan de forma iterativa usando la siguiente regla:

$$d^k = \begin{cases} -\nabla f(x^k) & si \quad k = 0 \\ -\nabla f(x^k) + \beta^{k-1}d^{k-1} & si \quad k \geq 1 \end{cases}$$

El escalar β^k se elige para reforzar la conjugación entre las direcciones. La opción más común es la propuesta de Fletcher-Reeves:

$$\beta_{FR} = \frac{||\nabla f(x^{k+1})||^2}{||\nabla f(x^k)||^2} \tag{5.8}$$

Sin embargo, la fórmula de Polak-Ribiére también puede usarse:

$$\beta_{PR} = \frac{\nabla f(x^{k+1})'(\nabla f(x^{k+1}) - \nabla f(x^k))}{||\nabla f(x^k)||^2} \tag{5.9}$$

Ambas fórmulas son iguales en el caso cuadrático y diferentes cuando se trate de cualquier otro caso.

El esquema del gradiente conjugado se presenta en Algoritmo 4.

Algorithm 4 Algoritmo del gradiente conjugado

1: **procedure** (Gradiente_Conjugado)
2: Establecer $x^0 \in \mathbb{R}^n$ y $k = 0$
3: **while** $\nabla f(x^k)! = 0$ **do**
4: Calcular β^{k-1} usando 5.8 o 5.9 y establecer la dirección usando

$$d^k = \begin{cases} -\nabla f(x^k) & si \quad k = 0 \\ -\nabla f(x^k) + \beta^{k-1} d^{k-1} & si \quad k \geq 1 \end{cases}$$

5: Encontrar α^k usando un algoritmo de búsqueda lineal que satisfaga las condiciones de Wolfe.
6: Establecer $x^{k+1} = x^k + \alpha^k d^k$ y $k = k + 1$.
7: **end while**
8: **end procedure**

Las condiciones de Wolfe mencionadas en 4 son las siguientes:

$$f(x^k + \alpha d^k) \leq f(x^k) + \gamma \alpha \nabla f(x^k)' d^k \tag{5.10}$$

que es la misma que se usó en la Búsqueda Lineal de Armijo, con la siguiente condición siendo más fuerte

$$|\nabla f(x^k + \alpha d^k)' d^k| \leq \beta |\nabla f(x^k)' d^k| \tag{5.11}$$

donde $\beta \in (\gamma, 1)$ y γ se encuentra en el mismo intervalo que antes.

La forma más simple de asegurar las propiedades de convergencia global en este método es aplicando reinicios periódicos junto con dirección de descenso más rápido. Aún así, el reinicio puede también suceder si alguno de los términos cuadráticos se pierden y pueden causar, o bien, que el método sea ineficiente, o bien, elecciones de caminos sin sentido.

El mecanismo de reinicio se realiza cada n iteraciones o si se da la siguiente condición:

$$|\nabla f(x^k)' \nabla f(x^{k+1})| > \delta ||\nabla f(x^{k-1})||^2$$

con $0 < \delta < 1$.

5.5. Métodos de Newton

Este método se considera uno de los algoritmos más potentes para resolver problemas de optimización sin restricciones. La aproximación cuadrática de la función objetivo en un vecindario de la solución actual, x^k, considerada es la siguiente

$$q^k(s) = \frac{1}{2}s'\nabla^2 f(x^k)s + \nabla f(x^k)'s + f(x^k)$$

donde necesitamos saber las derivadas de primer y segundo orden de la función objetivo en la iteración número k. Este algoritmo también necesita calcular una dirección, d_N, y en ese caso, se obtiene como un punto estacionario de la aproximación anterior y es la solución del sistema dado por

$$\nabla^2 f(x^k)d_N = -\nabla f(x^k) \tag{5.12}$$

Por lo tanto, la matriz $\nabla^2 f(x^k)d_N$ es no singular, tiene una inversa y la dirección es $d_N = -(\nabla^2 f(x^k))^{-1}\nabla f(x^k)$.

El esquema básico algorítmico está definido por la iteración

$$x^{k+1} = x^k - (\nabla^2 f(x^k))^{-1}\nabla f(x^k) \tag{5.13}$$

La calidad de este método se debe al hecho de que si el punto de partida x^0 está próximo a la solución x^*, entonces la secuencia de puntos generado por la ecuación anterior converge a x^* de forma superlineal o cuadrática (si la Hessiana es continuamente Lipschitziana en un vecindario de la solución).

Sin embargo, este método tiene algunas desventajas. Una de ellas es la singularidad de la matriz $\nabla^2 f$ porque en el caso de ser singular, el método no puede definirse. Otra desventaja está relacionada con el punto inicial, x^0. Puede ser tal que la secuencia generada por 5.13 no converge, pero puede ocurrir la convergencia a un punto máximo.

Debido a estos hechos, el método de Newton necesita algunos cambios para asegurar la convergencia global a la solución. Un método de Newton convergente debería generar una secuencia de puntos $\{x^k\}$ con las siguientes características:

- Admite un punto límite.
- Cualquier otro punto límite pertenece a $\mathcal{L}$ y es un punto estacionario de f.
- Ningún punto límite es un punto máximo de f.
- Si x^* es un punto límite de $\{x^k\}$ y $\nabla^2 f(x^*)$ es definida positiva, entonces la convergencia es, al menos, superlineal.

Hay dos enfoques para diseñar un método de Newton convergente globalmente: un enfoque con búsqueda lineal y un enfoque con regiones de confianza.

5.5.1. Modificaciones de Búsqueda Lineal en el Método de Newton

la adaptación del método a este enfoque es el control del tamaño de paso junto con d_N tal que 5.13 se convierte en

$$x^{k+1} = x^k - \alpha^k[\nabla^2 f(x^k)]^{-1}\nabla f(x^k) \tag{5.14}$$

donde α^k se elige con una buena búsqueda local. Un ejemplo puede ser inicializar $\Delta^k = 1$ en Algoritmo 1. Adicionalmente a este cambio, la dirección d_N puede ser perturbada para asegurar la convergencia global del algoritmo. La forma más fácil de realizar este cambio es usar la dirección del descenso más rápido siempre cuando d_N^k no satisface alguna de las condiciones de convergencia.

Un posible esquema de dicho algoritmo se presenta en Algoritmo 5.

En el tercer paso, se toma la dirección del descenso más rápido si $\nabla f(x^k)' d_N^k \geq 0$. Otra posible modificación del método de Newton podría ser aquella que toma la dirección de la curvatura negativa, es decir, $d^k = -d_N^k$. Esta modificación se puede hacer si las siguientes dos condiciones se cumplen:

1. $|\nabla f(x^k)' d^k| \geq c_1 ||\nabla f(x^k)||^q$
2. $||d^k||^p \leq c_2 ||\nabla f(x^k)||$

Algorithm 5 Método de Newton con Búsqueda Lineal

procedure (ABLNewton) $(c_1 > 0, c_2 > 0, p \geq 2, q \geq 3)$
  Establecer $x^0 \in \mathbb{R}^n$ y $k = 0$
  while $\nabla f(x^k)! = 0$ **do**
    if $\exists d_N^k$ solución de $\nabla^2 f(x^k) d_N^k = -\nabla f(x^k)$ y satisface

$$\nabla f(x^k)' d_N^k \leq -c_1 ||\nabla f(x^k)||^q, ||d_N^k||^p \leq c_2 ||\nabla f(x^k)||$$

then
      Establecer la dirección $d^k = d_N^k$
    else
      $d^k = -\nabla f(x^k)$
    end if
    Encontrar α^k usando 1
    Establecer $x^{k+1} = x^k + \alpha^k d^k$ y, después, $k = k + 1$
  end while
end procedure

Una segunda modificación es perturbar la matriz Hessiana con una matriz definida positiva Y^k y ahora la solución provendría de resolver el sistema $(\nabla^2 f(x^k) + Y^k)d = -\nabla f(x^k)$.

Las modificaciones comunes de los métodos de Newton se basan en el decremento monótono de los valores de la función objetivo. Con estos cambios la región de convergencia del método puede aumentar más de lo esperado, pero una secuencia convergente en este conjunto puede no ser una secuencia monótonamente descendente de los valores de la función objetivo.

La siguiente modificación está basada en las condiciones de búsqueda lineal y mantiene la propiedad de convergencia global. En esta modificación, usaremos una regla no monótona. Por lo tanto, el método obtenido se presenta en Algoritmo 6.

Los métodos de Búsqueda Lineal estudiados hasta ahora convergen a puntos satisfaciendo solo las condiciones de optimalidad de primer orden necesarias de la ecuación 5.1. Esto se debe a que el método de Newton no explota toda la información obtenida en la segunda derivada. Es posible obtener una convergencia más fuerte si usamos el par de direcciones (d^k, s^k) y una búsqueda curvilinear, es decir,

$$x^{k+1} = x^k + \alpha^k d^k + (\alpha^k)^{\frac{1}{2}} s^k \tag{5.15}$$

Algorithm 6 Método de Newton no monótono

procedure (NewtonNM) ($c_1 > 0, c_2 > 0, p \geq 2, q \geq 3$ y M entero)
  Establecer $x^0 \in \mathbb{R}^n$ y $k = 0$
  while $\nabla f(x^k)! = 0$ **do**
    if $\exists d_N^k$ solución de $\nabla^2 f(x^k) d_N^k = -\nabla f(x^k)$ y satisface

$$\nabla f(x^k)' d_N^k \leq -c_1 ||\nabla f(x^k)||^q, ||d_N^k||^p \leq c_2 ||\nabla f(x^k)||$$

then
      Establecer la dirección $d^k = d_N^k$
    else
      $d^k = -\nabla f(x^k)$
    end if
    Encontrar α^k usando 1, tal que

$$f(x^k + \alpha d^k) \leq \max_{0 \leq j \leq J} \{f(x^{k-j})\} + \gamma \alpha \nabla f(x^k)' d^k$$

con $J = m(k)$
    Establecer $x^{k+1} = x^k + \alpha^k d^k$ y, después, $k = k + 1$
    Establecer $m(k) = \min\{m(k-1) + 1, M\}$
  end while
end procedure

donde d^k es una dirección del método de Newton y s^k es una dirección que incluye información de curvatura negativa con respecto a $\nabla^2 f(x^k)$. Con esta idea y algunas modificaciones a la Búsqueda Lineal de Armijo, se pueden crear algoritmos globalmente convergente que además puedan satisfacer las condiciones de optimalidad de segundo orden necesarias para la ecuación 5.1.

5.5.2. Modificaciones de Regiones de Confianza en el Método de Newton

Este tipo de algoritmos tienen su iteración principal como muestra la siguiente ecuación

$$x^{k+1} = x^k + s^k$$

donde el paso s^k se obtiene minimizando la forma cuadrática q^k de la función objetivo en una región de confianza del espacio $\mathbb{R}^n$. La región de confianza se define como una norma l_p del paso s. Lo más común es elegir la norma Euclidiana, con la cual, en cada iteración, s^k es la solución de

$$\min_{s \in \mathbb{R}^n} \frac{1}{2} s' \nabla^2 f(x^k) s + \nabla f(x^k)' s$$

donde $||s||^2 \leq (a^k)^2$ con a siendo el radio de la región de confianza. Otra opción consistiría en realizar un cambio de escala a la condición previa de la siguiente forma: $||D^k s||^2 \leq (a^l)^2$. Por simplicidad, asumiremos que la matriz $D^k = I$, es decir, la matriz identidad en el espacio adecuado.

Estos algoritmos se caracterizan por la siguiente idea: cuando la matriz $\nabla^2 f(x^k)$ es definida positiva, entonces el radio a^k tiene que ser lo suficientemente grande que el minimizador de 5.5.2 no tenga restricciones y el paso dado por Newton sea un entero. Además, a^k se actualiza en cada iteración y su instrucción de actualización depende de la proporción ρ^k entre la reducción de los valores de la función objetivo $f(x^k) - f(x^{k+1})$ y la reducción esperada $f(x^k) - q^k(s^k)$.

El siguiente algoritmo (Algoritmo 7) presenta las ideas anteriores y también garantiza la satisfacción de las condiciones de optimalidad necesarias para 5.1 en su tercer paso.

Si f es además continuamente 2-diferenciable, entonces la secuencia del algoritmo anterior, $\{x^k\}$, tiene un punto límite que satisface las condiciones de optimalidad necesarias de primer y segundo orden para la ecuación 5.1. Además, si $\{x^k\}$ converge a un punto en el que la matriz Hessiana $\nabla^2 f$ es definida positiva, entonces el orden de convergencia es superlineal.

El esfuerzo computacional de este algoritmo es el subproblema de las regiones de confianza. Debido a este hecho, se han ido desarrollando cada vez más algoritmos para resolverlo. Sin embargo, no necesitamos una solución exacta para esta ecuación. Para probar que la convergencia global del algoritmo es suficiente verificar que el valor $q^k(s^k)$ es menor que el valor en un punto de Cauchy, que es el punto que minimiza el modelo cuadrático en la región de confianza.

Algorithm 7 Método de Newton basado en Regiones de Confianza

procedure (NewtonRegionConfianza) $(0 < \gamma_1 \leq \gamma_2 < 1, 0 \delta_1 < 1 \leq \delta_2)$
  Establecer $x^0 \in \mathbb{R}^n$, $k = 0$ y $a^0 = 0$.
  Encontrar $s^k = arg\,\text{mín}_{\{} ||s|| \leq a^k\} q^k(s)$
  if $f(x^k) == q^k(s^k)$ **then**
    Parar
  else
    Calcular la proporción

$$\rho^k = \frac{f(x^k) - f(x^k + s^k)}{f(x^k) - q^k(s^k)}$$

    if $\rho^k \geq \gamma_1$ **then**
      Actualizar $x^{k+1} = x^k + s^k$
    else
      Actualizar $x^{k+1} = x^k$
    end if
    Actualizar el radio a^k

$$a^{k+1} = \begin{cases} \delta_1 a^k & si \quad \rho^k < \gamma_1 \\ a^k & si \quad \rho^k \in\in [\gamma_1, \gamma_2] \\ \delta_2 a^k & si \quad \rho^k > \gamma_2 \end{cases}$$

  y establecer $k = k + 1$, entonces volver al paso 3.
  end if
end procedure

5.5.3. Métodos de Newton Truncados

Los métodos de Newton necesitan calcular la solución de un sistema lineal de ecuaciones en cada iteración. Si echamos un vistazo a problemas a gran escala, resolver este sistema en cada iteración puede resultar demasiada carga computacionalmente hablando. Además, la solución exacta cuando x^k está lejos de una solución y $||\nabla f(x^k)||$ es grande no es necesaria. Debido a este hecho, se han propuesto numerosos métodos que calculan una solución aproximada a este sistema con un orden de convergencia bueno, es decir, si $\widetilde{d_N^k}$ es una solución aproximada del sistema, entonces la medida

de precisión es dada por el residual de la ecuación de Newton, que sería

$$r^k = \nabla^2 f(x^k)\widetilde{d_N^k} + \nabla f(x^k)$$

Si podemos controlar este residual, entonces podemos probar la convergencia superlineal.

Proposición 5.3 *Si* $\{x^k\}$ *converege a una solución y si*

$$\lim_{k\to\infty} \frac{||r^k||}{||\nabla f(x^k)||} = 0$$

entonces $\{x^k\}$ *converge de forma superlineal.*

Estos métodos solo requieren de operaciones matriciales-vectoriales, por lo que son adecuados para problemas a gran escala. Al siguiente algoritmo se le conoce como el método de Newton truncado y se necesita que la matriz Hessiana sea definida positiva. Este método se obtiene aplicando un esquema de gradiente conjugado para encontrar una solución óptima del sistema 5.12.

Generaremos los vectores d_N^i, que se van a ir aproximando a la dirección d_N^k y se detendrá cuando ocurra uno de los siguientes casos:

- El residual r^i verifica que $||r^i|| \leq \epsilon^k$, para $e^k > 0$.
- Si se ha encontrado una dirección de curvatura negativa, es decir, la dirección conjugada s^i verifica que $(s^i)'\nabla^2 f(x^k)s^i \leq 0$.

Este algoritmo tiene el mismo resultado para convergencia que el método de Newton. Por simplicidad, eliminaremos las dependencias en la iteración k y estableceremos $H = \nabla^2 f(x^k)$ y $g = \nabla f(x^k)$. Este método genera las direcciones conjugadas s^i y los vectores p^i que aproximan la solución del sistema de Newton $\widetilde{d_N^k}$. Esto se puede encontrar representado en el Algoritmo 8.

Algorithm 8 Algoritmo de Newton Truncado

1: **procedure** (NewtonRegionConfianza) (k, H, g y $\eta > 0$ escalar)
2: Establecer $i = 0, p^0 = 0, r^0 = -g, s^0 = r^0$ y

$$\epsilon = \eta||g|| \text{ mín}\left\{\frac{1}{k+1}, ||g||\right\}$$

3: **while** **do**
4: **end while**
5:

$$a_N = \begin{cases} \delta_1 a^k & si \quad \rho^k < \gamma_1 \\ a^k & si \quad \rho^k \in\in [\gamma_1, \gamma_2] \\ \delta_2 a^k & si \quad \rho^k > \gamma_2 \end{cases}$$

6: **end procedure**

Si aplicamos este algoritmo para encontrar una solución aproximada al sistema en el cuarto paso de 5 o 6, entonces obtendremos la versión monótona truncada del método de Newton.

5.5.4. Métodos Quasi-Newton

Estos métodos han sido introducidos con el objetivo de diseñar algoritmos eficientes que no necesiten la evaluación de derivadas de segundo orden. Por tanto, han establecido d^k como la solución de

$$B^k d = -\nabla f(x^k) \tag{5.16}$$

donde B^k es una matriz definida positiva simétrica de tamaño $n \times n$ que se ajusta de forma iterativa para que la dirección d^k tienda a aproximar la dirección del método de Newton. A la fórmula anterior se le conoce como **fórmula quasi-Newton** y su inversa es

$$d^k = -H^k \nabla f(x^k) \tag{5.17}$$

Ambas matrices se modifican en cada iteración como una correción de la anterior, es decir, $B^{k+1} = B^k + \Delta B^k$, y lo mismo pasa para H^k. Definimos las siguientes dos cantidades:

$$\delta^k = x^{k+1} - x^k \qquad \gamma^k = \nabla f(x^{k+1}) - \nabla f(x^k)$$

En caso de que f sea una función cuadrática, entonces la ecuación quasi-Newton sería

$$\nabla^2 f(x^k)\delta^k = \gamma^k \tag{5.18}$$

por lo tanto, ΔB^k (lo mismo para ΔH^k) se elige de la siguiente forma

$$(B^k + \Delta B^k)\delta^k = \gamma^k \tag{5.19}$$

La regla de actualización de H^k está dada por

$$\Delta H = \frac{\delta^k(\delta^k)'}{(\delta^k)'\gamma^k} - \frac{H^k\gamma^k(H^k\gamma^k)'}{(\gamma^k)'H^k\gamma^k} + c(\gamma^k)'H^k\gamma^k v^k(v^k)' \tag{5.20}$$

donde

$$v^k = \frac{\delta^k}{(\delta^k)'\gamma^k} - \frac{H^k\gamma^k}{(\gamma^k)'H^k\gamma^k}$$

y $c > 0$ es un escalar. El algoritmo que vamos a mostrar ahora fue creado por Broyden, Flecher, Goldfarb y Shanno. Es un método de búsqueda lineal en el que el tamaño de paso es α^k se obtiene mediante un algoritmo de búsqueda lineal. El esquema del algoritmo se presenta en Algoritmo 9.

Distinguimos dos casos en relación a las propiedades de convergencia: caso convexo y caso no convexo. Cuando no tenemos convexidad, si existe una constante ρ tal que para cada k se verifica la siguiente condición

$$\frac{||\gamma^k||^2}{(\gamma^k)'\delta^k} \leq \rho \tag{5.22}$$

entonces la secuencia de puntos generada por el algoritmo satisface

$$\lim_{k \to \infty} inf||\nabla f(x^k)|| = 0$$

que es la condición débil de 4.5. Para el caso convexo, la desigualdad anterior se mantiene. El siguiente resultado nos dará información sobre el orden de convergencia de este algoritmo.

Algorithm 9 Algoritmo BFGS Inverso Quasi-Newton

procedure (InversaBFGSQuasiNewton)
  Establecer $x_0 \in \mathbb{R}^n, H^0 = I$ y $k = 0$
  while $\nabla f(x^k)! = 0$ **do**
    Establecer la dirección $d^k = -H^k \nabla f(x^k)$
    Encontrar α^k mediante una búsqueda lineal que satisfaga las condiciones de Wolfe 5.10 y 5.11
    Actualizar

$$\begin{aligned} x^{k+1} &= x^k + \alpha^k d^k \\ H^{k+1} &= H^k + \Delta H^k \end{aligned} \tag{5.21}$$

    con ΔH^k dada por la regla 5.20 con $c = 1$.
    Establecer $k = k + 1$
  end while
end procedure

Proposición 5.4 *Sea $\{B^k\}$ una secuencia de matrices no singulares y sea $\{x^k\}$ la secuencia dada por*

$$x^{k+1} = x^k - (B^k)^{-1} \nabla f(x^k)$$

También supondremos que $\{x^k\}$ converge a un punto x^* donde $\nabla^2 f(x^*)$ es también no singular. Entonces, la secuencia $\{x^k\}$ converge de forma superlineal a $x^* \Leftrightarrow$

$$\lim_{k \to \infty} \frac{||[B^k - \nabla^2 f(x^*)](x^{k+1} - x^k)||}{||x^{k+1} - x^k||} = 0$$

Este algoritmo está pensado para problemas de pequeña escala, ya que para los de gran escala, el almacenar una matriz como serían B^k o H^k ocasionaría problemas de almacenamiento. Por lo tanto, para esos problemas la información se obtiene de las últimas iteraciones.

5.5.5. Métodos sin derivadas

Estos métodos no calculan explícitamente las derivadas de f. Son adecuados para cuando, o bien, el gradiente de la función objetivo no puede ser calculado, o bien, cuando es computacionalmente costoso. Sin embargo, si queremos probar las propiedades de convergencia, necesitaremos suponer que f es continuamente diferenciable.

De entre todos estos algoritmos, los dos más importantes son los algoritmos de búsqueda de patrones (PSA, *pattern-search algorithm*) y los algoritmos de búsqueda lineal sin derivadas (DFLSA, *derivative-free line-search algorithm*). Estos algoritmos son similares, y su diferencia reside en las suposiciones que hacen sobre el conjunto de direcciones y en la regla que usan para encontrar el tamaño de paso junto con las direcciones. Denotamos $\mathcal{D}^k = \{d^1, ..., d^r\}$ con $r \geq n + 1$ como el conjunto de direcciones y suponemos que son unitarias.

También asumimos el siguiente hecho para los PSA: las direcciones $d^j \in \mathcal{D}^k$ son la j-ésima columna de la matriz $B\Gamma^k$ con B una matriz no singular con coeficientes reales de tamaño n y $\Gamma^k \in \mathcal{M} \subset \mathbb{Z}^{n \times r}$, donde $\mathcal{M}$ es un conjunto finito de matrices integrales tales que su rango coincide con el número de filas que tienen (*full row-rank*).

Esta suposición nos da una idea para entender que el PSA itera sobre x^{k+1} en una red (*lattice*)

racional centrada en x^k. Sea $\mathcal{P}^k$ el conjunto de candidatos para la siguiente iteracion, es decir,

$$\mathcal{P} = \{x^{k+1} : x^k + \alpha^k d^j, d^j \in \mathcal{D}^k\}$$

En este conjunto se conoce el patrón y se elige el tamaño de paso para preservar la estructura algebraica en la siguiente iteración, por lo que tenemos $f(x^k + s^k) < f(x^k)$ con $s^k = \alpha^k d^j$.

Para DLFSA haremos la siguiente suposición: las direcciones $d^j \in \mathcal{D}^k$ son la j-ésima columna de una matriz B^k de tamaño $n \times r$ con rango n. En este caso, no hay suposiciones adicionales y solo tiene que verificar la reducción de la función objetivo.

Los dos siguientes algoritmos muestran una versión del PSA (Algoritmo 10) y del DFLSA (Algoritmo 11).

Algorithm 10 Algoritmo de Búsqueda de Patrones

procedure (PSA) ($\mathcal{D}^{\|}$ satisfaciendo la suposición previa, $\tau \in \{1, 2\}, \theta = \frac{1}{2}$)
  Establecer $x_0 \in \mathbb{R}^n, \Delta^0 > 0$ y $k = 0$
  Comprobar la convergencia
  if $\exists j \in \{1, \ldots, r\}$:

$$f(x^k + \alpha^k d^j) < f(x^k), \alpha^k = \Delta^k$$

  then
    Establecer $x^{k+1} = x^k$, $\Delta k + 1 = \tau \Delta^k$, $k = k + 1$ e ir al paso 3.
  else
    Establecer $x^{k+1} = x^k$, $\Delta k + 1 = \theta \Delta^k$, $k = k + 1$
  end if
end procedure

Algorithm 11 Algoritmo de Búsqueda Lineal sin Derivadas

procedure (DFLSA) ($\mathcal{D}^{\|}$ satisfaciendo la suposición previa, $\gamma > 0$, $\theta \in (0, 1)$)
  Establecer $x_0 \in \mathbb{R}^n, \Delta^0 > 0$ y $k = 0$
  Comprobar la convergencia
  if $\exists j \in \{1, \ldots, r\}$:

$$f(x^k + \alpha^k d^j) \leq f(x^k) - \gamma(\alpha^k)^2, \alpha^k \geq \Delta^k$$

  then
    Establecer $x^{k+1} = x^k$, $\Delta k + 1 = \alpha^k$, $k = k + 1$ e ir al paso 3.
  else
    Establecer $x^{k+1} = x^k$, $\Delta k + 1 = \theta \Delta^k$, $k = k + 1$
  end if
end procedure

Ambos algoritmos generan una secuencia que verifica la condición débil de convergencia de 4.5, es decir,

$$\lim_{k \to \infty} inf ||\nabla f(x^k)|| = 0$$

En este caso, tomamos un índice, j, en PSA tal que

$$f(x^k + \alpha^k d^j) = \min_{i : d^i \in \mathcal{D}^k} f(x^k + \alpha^k d^i) < f(x^k)$$

también mantenga la premisa de 4.5. Esto también ocurre en DFLSA, porque solo tenemos que tomar un tamaño de paso, α^k, tal que

$$f\left(x^k + \frac{\alpha^k}{\delta}d^k\right) \geq \text{máx}\left\{f(x^k + \alpha^k d^k), f(x^k) - \gamma\left(\frac{\alpha^k}{\delta}\right)^2\right\}, \delta \in (0,1)$$

Ambos esquemas encuentran un tamaño de paso que les permita comprobar la convergencia del algoritmo. En estos algoritmos no necesitamos tener el gradiente de f, por lo que tenemos que comprobar la siguiente condición:

$$\sqrt{\frac{\sum_{i=1}^{n+1}(f(x^i) - \overline{f})^2}{n+1}} \leq tolerancia \tag{5.23}$$

donde $\overline{f} = \dfrac{1}{n+1}\sum_{i=1}^{n+1} f(x^i)$ y $\{x^i : i = 1, \ldots, n+1\}$ incluye el punto actual y los n puntos anteriormente generados junto con las n direcciones.

Como resultado de esta teoría, tenemos algoritmos que nos permitan generar secuencias e puntos en $\mathbb{R}^n$ que converjan a los puntos óptimos y, bajo ciertas circunstancias, pueden ser el óptimo global. Estos métodos son el principio de la gran cantidad de algoritmos presentados en la literatura. Estos algoritmos proponen formas de alcanzar un óptimo global de funciones basadas en una idea inspirada en la naturaleza.

Incluso si la gran cantidad de algoritmos que presentaremos en las próximas secciones, el diseño de cada uno es similar a cómo esta teoría propone el movimiento, es decir, metaheurísticas basando su movimiento en una dirección dada por un vector y el criterio de parada basado en condiciones que normalmente presentan la optimalidad del mejor individuo de la población.

Capítulo 6: Tests estadísticos

6.1. Comparaciones por parejas

Las comparaciones por parejas son el tipo de tests estadísticos más simples que un investigador puede aplicar en el contexto de un estudio experimental. Estos tests son útiles para comparar el rendimiento de dos algoritmos cuando se aplican a un conjunto de problemas común. En el análisis de múltiples problemas, se requiere un valor por cada par de algoritmo-problema (frecuentemente un valor medio de varias ejecuciones).

Dentro de los tests estadísticos debemos diferenciar entre test paramétricos y no paramétricos. La diferencia fundamental entre ambos tests está basada en el conocimiento o desconocimiento de la distribución de probabilidad de la variable que se pretende estudiar. De entre los tests paramétricos es importante destacar el test de *t-student* (usado para comparar las medias de dos grupos y determinar si las diferencias entre ellas probablemente provengan de la aleatoriedad) y ANOVA (generaliza el test *t-student*, ya que permite estudiar si las medias de dos o más poblaciones son iguales). De entre los test no paramétricos, debemos destacar el test de Wilcoxon y los procedimientos *post-hoc*; ambos se explicarán en detalle más adelante.

Los tests paramétricos se suelen usar en el análisis de experimentos en inteligencia computacional. Desafortunadamente, se basan en suposiciones que con casi toda seguridad no se cumplen al analizar el rendimiento de algoritmos estocásticos basados en inteligencia computacional [38]. Estas suposiciones se conocen como independencia, normalidad y homocedasticidad. Para superar este problema, nos centraremos en los procedimientos estadísticos no paramétricos, que aportan al investigador una herramienta práctica a usar cuando las suposiciones previas no pueden ser satisfechas, especialmente en el análisis de múltiples problemas.

En esta sección, nos centraremos en primer lugar en un procedimiento fácil y potente, que puede proporcionar una primera impresión sobre la comparación: el Test de Signos. Entonces, introduciremos el uso del test de posiciones con signos de Wilcoxon, como un ejemplo de un test no paramétrico para comparaciones estadísticas por parejas.

6.1.1. Test de signos

Una forma popular de comparar el rendimiento general de los algoritmos es contar el número de casos en los que un algoritmo es el ganador (tiene el mejor rendimiento) de entre todos. Algunos autores también utilizan este conteo en estadística inferencial, con un test conocido como Test de signos. Si amso algoritmos comparados son, como se asume en la hipótesis nula, equivalentes, cada uno debería ganar en aproximadamente $n/2$ de n problemas.

El número de victorias se distribuye de acuerdo a una distribución binomial: para un mayor número de casos, el número de victorias está bajo la hipótesis nula de acuerdo a $n(n/2, \sqrt{n}/2)$, lo

que permite el uso del Z-test: si el número de victorias es, al menos, $n/2 + 1.96 \cdot \sqrt{n}/2$, entonces el algoritmo es significativamente mejor con $p < 0.05$.

6.1.2. Test de posiciones con signos de Wilcoxon

Este test se utiliza para responder a la siguiente pregunta: ¿Dos muestras representan 2 poblaciones diferentes? Es un procedimiento no paramétrico utilizado en situaciones de testeo de hipótesis, involucrando un deiseño con dos muestras. Esto es análogo al t-test por parejas en procedimientos estadísticos no paramétricos; por lo tanto, es un test por parejas cuyo objetivo es detectar diferencias significativas entre dos muestras,es decir, el comportamiento de dos algoritmos.

Definición 6.1 *Test de Wilcoxon Sea d_i la diferencia entre la puntuación del rendimiento de los algoritmos en el problema i-ésimo de entre n problemas (si esta puntuación se pueden representar en distintas posiciones, pueden ser normalizados en el intervalo [0,1], con el objetivo de no darle prioridad a ningún problema). Las diferencias se ordenan según sus valores absolutos; en caso de empates, se puede aplicar alguno de los métodos disponibles en la literatura (ignorar empates, asignar la posición más elevada, calcular todas las posibilidades y hacer la media de los resultados obtenidos para cada aplicación del test...), aunque recomendamos el uso de hacer la media de las posiciones (por ejemplo, si dos diferencias están empatadas en la asignación de posiciones 1 y 2, entonces se le asignará la posición 1.5 a ambas).*

Sea $\mathbb{R}^+$ la suma de las posiciones para los problemas tales que el primer algoritmo mejoró al segundo, y $\mathbb{R}^-$ la suma de las posiciones del caso contrario. Las posiciones de $d_i = 0$ se dividen a partes iguales entre ambas sumas; si hay un número impar de estas, una de ellas se ignora:

$$\begin{aligned} R^+ &= \sum_{d_i>0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i) \\ R^- &= \sum_{d_i<0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i) \end{aligned} \tag{6.1}$$

Sea $T = \text{mín}(R^+, R^-)$. si T es menor o igual que el valor de la distribución de Wilcoxon para n grados de libertad, se rechaza la hipótesis nula de igualdad; esto significa que dicho algoritmo supera al otro, con el p-valor asociado.

El test de Wilcoxon es más robusto que el t-test. Asume conmensurabilidad de diferencias, pero solo de forma cualitativa: mayores diferencias contarán más, lo que es probablemente lo que se desea, pero las magnitudes absolutas se ignoran. Desde un punto de vista estadístico, este test es más seguro ya que no asume una distribución normal. Además, los valores atípicos (*outliers*) tienen menos efecto en el test de Wilcoxon que en el t-test.

6.2. Múltiples comparaciones con un método de control

Una de las situaciones más frecuentes donde el uso de proceedimientos estadísticos es necesario es el análisis conjunto de resultados obtenidos por distintos algoritmos. Los grupos de diferencias entre estos métodos (también llamados bloques) se suelen asociar con los problemas con los que se trabaja en el estudio experimental. Cuando nos referimos a test de comparaciones múltiples, un bloque se compone de tres o más sujetos o resultados, cada uno de ellos correspondiendo a la evaluación de rendimiento del algoritmo sobre el problema.

En el análisis por parejas, si se intenta extraer una conclusión que involucre más de una comparación por parejas, obtendremos un error acumulado procedente de su combinación. En términos estadísticos, estamos perdiendo el control en el error producido por familia (*Family-Wise Error Rate*, FWER), definido como la probabilidad realizar descubrimientos falsos de entre todas las hipótesis cuando se usan los test múltiples por parejas. La verdadera significación estadística de combinar comparaciones por parejas viene dada por:

$$\begin{aligned} p &= P(\text{Rechazar } H_0|H_0 \text{ cierto}) = 1 - P(\text{Aceptar } H_0|H_0 \text{ cierto}) \\ &= 1 - P(\text{Aceptar } A_k = A_i, i = 1, ..., k-1|H_0 \text{ cierto}) \\ &= 1 - \prod_{i=1}^{k-1} P(\text{Acceptar} A_k = A_i|H_0 \text{ cierto}) \\ &= 1 - \prod_{i=1}^{k-1} [1 - P(\text{Rechazar } A_k = A_i|H_0 \text{ cierto})] \\ &= 1 - \prod_{i=1}^{k-1} (1 - p_{H_i}) \end{aligned}$$

Por lo tanto, un test de comparación por parejas, tal como el test de Wilcoxon, no debería usarse para realizar varias comparaciones involucrando un conjunto de algoritmos, porque el FWER no está controlado.

Esta sección está dedicada a describir el uso de diversos procedimientos para comparaciones múltiples considerando un método de control. En este sentido, un método de control puede ser definido como el algoritmo más interesante para el investigador del estudio experimental (normalmente este suele ser la nueva propuesta). Por lo tanto, su rendimiento será contrastado con el resto de algoritmos del estudio.

6.2.1. Test múltiple de signos

Dado un algoritmo de control etiquetado, el test de signos para múltiples comparaciones nos permite destacar aquellos cuyos rendimientos son estatísticamente distintos cuando comparados con el algoritmo de control. Este procedimiento es el siguiente:

1. Representaremos por $x_{i,1}$ y $x_{i,j}$ los rendimientos del algoritmo de control y el algoritmo j-ésimo en el i-ésimo problema.

2. Calculamos las diferencias de signos $d_{i,j} = x_{i,j} - x_{i,1}$. Esto es, emparejar cada rendimiento con el control y, en cada problema, restar el rendimiento del algoritmo de control al rendimiento del j-ésimo algoritmo.

3. Sea r_j el número de diferencias, $d_{i,j}$ que tienen el signo con la menor frecuencia de aparición con un emparejamiento de un algoritmo con el de control.

4. Sea M_1 la respuesta media de una muestra de resultados del algoritmo de control y M_j la respuesta media de una muestra de los resultados del algoritmo j-ésimos. Aplicaremos una de las siguientes reglas de decisión:

   - Para contrastar $H_0 : M_j \geq M_1$ frente $H_1 : M_j < M_1$, rechazaremos H_0 si el número de símbolos negativos es menor o igual al valor crítico de R_j para $k - 1$ (número de

algoritmos excluyendo el de control), n (número de problemas), y la tasa de error del experimento elegida.

- Para contrastar $H_0 : M_j \leq M_1$ frente $H_1 : M_j > M_1$, rechazaremos H_0 si el número de símbolos positivos es menor o igual al valor crítico de R_j para $k-1$, n, y la tasa de error del experimento elegida.

6.2.2. Los test de Friedman, Posiciones Alineadas de Friedman y Quade

El test de Friedman es un análogo no paramétrico del análisis paramétrico ANOVA. Puede usarse para responder a la siguiente pregunta: en un conjunto de $k \geq 2$ muestras de n datos cada una, ¿hay al menos dos muestras que representen poblaciones con distintas medias? El test de Friedman es análogo a repetidas medidas ANOVA en procedimientos estadísticos no paramétricos; por lo tanto, es un test de múltiples comparaciones cuyo objetivo es detectar diferencias significativas entre el comportamiento de dos o más algoritmos.

La hipótesis nula del test de Friedman establece la igualdad de medias entre las poblaciones. La hipótesis alternativa se define como la negación de la hipótesis nula, por lo que es no direccional.

El primer paso para calcular el test estadístico es convertir los resultados originales en posiciones. Esto se calcula utilizando el siguiente procedimiento:

1. Reunir los resultados observados para cada par algoritmo-problema.
2. Por cada problema i, ordena los valores de 1 (mejor resultado) a k (peor resultado). Denotaremos estas posiciones como r_i^j $(1 \leq k)$.
3. Para cada algoritmo j, hacemos la media de las posiciones obtenidas en todos los problemas para calcular la posición final $R_j = \dfrac{1}{n}\sum_i r_i^j$.

Por ello, ordena los algoritmos de cada problema por separado; el algoritmo con mejor rendimiento debería obtener la posición 1, el segundo la posición 2, etc. En caso de empate, se recomienda calcular la media de las posiciones. Bajo la hipótesis nula, que establece que todos los algoritmos se comportan de forma similar (por lo tanto sus posiciones R_j deberían ser iguales) la estadística de Friedman F_j se puede calcular como

$$F_j = \frac{12n}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{6.2}$$

que se distribuye según una distribución χ^2 con $k-1$ grados de libertad, donde n y k son lo suficientemente grandes (normalmente, $n > 10$ y $k > 5$). Para menor número de algoritmos y problemas, los valores críticos ya han sido calculados.

Iman y Davenport propusieron una derivación del estadístico de Friedman en tnato que esta última métrica suele producir un efecto conservativo no deseado. El estadístico propuesto es

$$F_{ID} = \frac{(n-1)\chi_F^2}{n(k-1)-\chi_F^2} \tag{6.3}$$

que se distribuye de acuerdo a una F-distribución con $k-1$ y $(k-1)(N-1)$ grados de libertad.

Un inconveniente del esquema de posiciones empleado por el test de Friedman es que solo permite comparaciones dentro del mismo conjunto. Cuando el número de algoritmos a comparar es

pequeño, esto no puede ser una desventaja, en tanto que las comparaciones en el mismo conjunto pueden no ser significantes.

En el método de posiciones alineadas para el test de Friedman, el valor de una localización se calcula como la media de los rendimientos alcanzados por todos los algoritmos y problemas.

Las diferencias resultantes, que mantienen sus identidades con respecto al problema y la combinación de algoritmos a los que pertenecen, son entonces ordenados de 1 a $k \cdot n$ relativos entre sí. Este esquema de posiciones es el mismo que se emplea en procedimientos de comparación múltiple que emplean muestras idependientes, como sería el test de Kruskal-Wallis. Las posiciones asignadas a las observaciones alineadas se llaman posiciones alineadas.

El test estadístico de Friedman de Posiciones Alineadas se puede definir como:

$$F_{AR} = \frac{(k-1)\left[\sum_{j=1}^{k} \hat{R}_j^2 - \frac{kn^2}{4}(kn+1)^2\right]}{\frac{kn(kn+1)(2kn+1)}{6} - \frac{1}{k}\sum_{i=1}^{n} \hat{R}_i^2} \tag{6.4}$$

donde $\hat{R}_i$ es igual a la posición total del problema i-ésimo y $\hat{R}_t$ es la posición total del j-ésimo algoritmo.

Finalmente, introduciremos un último test para realizar comparaciones múltiples: el test de Quade. Este test, en contraste al test de Friedman, toma en cuenta el hecho de que algunos problemas son más difíciles o que las diferencias registradas en la ejecución de varios algoritmos sobre ellos son mayores (el test de Friedman considera todos los problemas iguales en términos de importancia). Por lo tanto, las posiciones se calculan calculados en cada problema pueden ser escalados dependiendo de las diferencias observadas en el rendimiento de los algoritmos, obteniendo, como resultado, un análisis de *ranking* con pesos de las muestras de resultados.

El procedimiento empieza encontrando las posiciones r_i^j de la misma forma que el test de Friedman. El siguiente paso requiere los valores originales del rendimiento de los algoritmos x_i^j. Las posiciones se asignan a los propios problemas de acuerdo con el tamaño del rango de la muestra en cada problema. El rango de la muestra dentro de los prolemas i es la diferencia entre las mayores y menores observaciones en dicho problema:

$$\text{Rango en un problema} : i = \max_j x_i^j - \min_j x_i^j \tag{6.5}$$

Obviamente, hay n rangos de muestra, uno por cada problema. Asignando la posición 1 al problema con el menor rango, la posición 2 al problema con el segundo menor rango... Se usan las posiciones medias en caso de empate. Sean $Q_1, Q_2, ..., Q_n$ las posiciones asignadas a los problemas $1, 2, ..., N$, respectivamente.

Finalmente, la posición de problema Q_i se multiplica por la diferencia entre la posición dentro del problema i, r_i^j, y la posición media dentro de los problemas, $(k+1)/2$, para obtener el producto S_i^j, donde

$$S_i^j = Q_i\left[r_i^j - \frac{k+1}{2}\right] \tag{6.6}$$

es un estadístico que representa el tamaño relativo de cada observación dentro de un problema, ajustado para reflejar la significancia relativa del problema en el que aparece. Además, definimos S_j como la suma de cada algoritmo, $S_j = \sum_{i=1}^{n} S_i^j$, para $j = 1, 2, ...k$.

Por conveniencia, y para establecer una relación con el test de Friedman, también usaremos *rankings* sin ajustes de medias,

$$W_i^j = Q_i \left[r_i^j \right] \tag{6.7}$$

y la posición media para el j-ésimo algoritmo, T_j, dado por

$$T_j = \frac{W_j}{n(n+1)/2} \tag{6.8}$$

donde $W_j = \sum_{i=1}^{n} W_i^j$, para $j = 1, 2, ..., k$.

Se deben establecer algunas definiciones para calcular el test estadístico, F_Q. Sean los términos A y B

$$\begin{aligned} A &= \frac{n(n+1)(2n+1)k(k+1)(k-1)}{72} \\ B &= \frac{1}{n} \sum_{j=1} k S_j^2 \end{aligned} \tag{6.9}$$

El test estadístico, F_Q, es

$$F_Q = \frac{(n-1)B}{A-B} \tag{6.10}$$

que se distribuye de acuerdo a la F-distribución con $k-1$ y $(k-1)(n-1)$ grados de libertad. Cuando se calcula el estadístico, nótese que, si $A = B$, debemos considerar que el punto se encuentra en la región crítica de la distribución estadística.

Para cada uno de estos test, una vez se hayan calculado los propios estadísticos, es posible calcular un p-valor mediante aproximaciones normales. Si hallamos la existencia de diferencias significantes, podemos proceder con el procesamiento *post-hoc* para caracterizar estas diferencias.

6.2.3. Procedimientos *post-hoc*

El principal inconveniente para los test de Friedman, Iman-Davenport, Friedman *Aligned* y Quade es que solo pueden detectar diferencias significativas sobre la comparación múltiple en su totalidad, siendo incapaz de establecer comparaciones propias entre algunos de los algoritmos considerados. Cuando el objetivo de la aplicación de tests múltiples es el realizar una comparación considerando un método de control y un conjunto de algoritmos, se puede definir una familia de hipótesis, relacionados con el método de control. Entonces, el uso de un test *post-hoc* puede conducir a la obtención de un p-valoro que determine el grado de rechazo de cada hipótesis.

Una familia de hipótesis es un conjunto de hipótesis de comparación lógicamente interrelacionadas tales que, en $1 \times N$ comparaciones, compara los $k-1$ alogoritmos del estudio (excluyendo el de control) con el método de control, mientras que en $N \times N$ comparaciones, considera las $k(k-1)/2$ posibles comparaciones entre algoritmos. Por lo tanto, la familia estará compuesta de $k-1$ o $k(k-1)/2$ hipótesis, respectivamente, tales que pueden ser ordenadas por su p-valor, desde el menor hasta el mayor.

El p-valor de cada hipótesis en la familia se puede obtener mediante una conversión de posiciones calculados por cada test mediante el uso de una aproximacicón normal. El test estadístico para comparar el i-ésimo y el j-ésimo algoritmo, z depende del procedimiento no paramétrico principal utilizado:

- **Test de Friedman**

$$z = (R_i - R_j)/\sqrt{\frac{k(k+1)}{6n}} \tag{6.11}$$

donde R_i y R_j son la posiciones medias por el test de Friedman de los algoritmos comparados.

- **Test de Friedman *Aligned***

$$z = (\hat{R}_i - \hat{R}_j)/\sqrt{\frac{k(n+1)}{6}} \tag{6.12}$$

donde $\hat{R}_i$ y $\hat{R}_j$ son las posiciones medias por el test de Posiciones Alineadas de Friedman para los algoritmos comparados.

- **Test de Quade**

$$z = (T_i - T_j)/\frac{k(k+1)(2n+1)(k-1)}{18n(n+1)} \tag{6.13}$$

donde $T_i = \frac{W_i}{n(n+1)/2}$, $T_j = \frac{W_j}{n(n+1)/2}$ y W_i y W_j son las posiciones sin el ajuste de medias por el test de Quade de los algoritmos comparados.

Sin embargo, estos p-valores no son adecuados para comparaciones múltiples. Cuando un p-valor es considerado en un test múltiple, refleja la posibilidad de error de una cierta comparación, pero no toma en consideración el resto de comparaciones pertenecientes a la familia. Si k algoritmos se comparan y en cada comparación el nivel de significancia es α, entonces en una comparación individual la probabilidad de no hacer un error de Tipo 1 (rechazar una hipótesis nula cierta) es (1-α), y la posibilidad de no hacer un error de Tipo 1 en $k-1$ comparaciones es $(1-\alpha)^{k-1}$. Por lo tanto, la posibilidad de hacer uno o más errores de tipo 1 es $1-(1-\alpha)^{k-1}$.

El z-valor en todos los casos se usa para encontrar la probabilidad correspondiente p-value de la tabla de la distribución normal $N(0,1)$, el cual es entonces comparado con un nivel de significancia, α, apropiado. Los tests *post-hoc* se diferencian en la forma en la que ajustan el valor de α para compensar las comparaciones múltiples.

A continuación, definiremos un conjunto de procedimientos *post-hoc* y explicaremos cómo calcular los APVs (*Analysis of Partial Variance*) dependiendo del procedimiento *post-hoc* usado en el análisis. La notación usada para describir el cálculo de las APVs es la siguiente:

- Índices i y j corresponden a una comparación en concreto o una hipótesis en una familia de hipótesis, de acuerdo con un orden incremental de sus p-valores. El índice i siempre se refiere a la hipótesis cuyo APV se está calculando y el índice j se refiere a otra hipótesis de en la familia.
- p_j es el p-valor obtenido por la j-ésimo hipótesis.

Los procedimientos del ajuste de p-valores pueden ser clasificados en distintos casos:

- **Un paso**:
  - El procedimiento de Bonferroni-Dunn: Ajusta los valores de α en un único paso mediante su división entre el número de comparaciones realizadas $(k-1)$. Este procedimiento es el más simple, pero el menos potente.
    Bonferroni APV_i: mín$\{v, 1\}$, donde $v = (k-1)p_i$.

- ***Step-down***:
  - El procedimiento de Holm: Ajusta el valor de α en una forma *step-down*. Sean $p_1, p_2, ..., p_{k-1}$ p-valores ordenados (de menor a mayor), tal que $p_1 \leq p_2 \leq \cdots \leq p_{k-1}$, y sean $H_1, H_2, ..., H_{k-1}$ las hipótesis correspondientes. El procedimiento de Holm rechaza H_1 a H_{i-1} si i es el menor entero tal que $p_i > \alpha(k-1)$. Este procedimiento empieza con el p-valor con mayor significancia. Si p_1 es menor que $\alpha/(k-1)$, se rechaza la hipótesis correspondiente y procedemos a comparar p_2 con $\alpha/(k-2)$. Si la segunda hipótesis es rechazada, el test procede con la tercera, y así sucesivamente. En cuanto alguna hipótesis nula no pueda ser rechazada, el resto de hipótesis se mantienen también.
    Holm APV_i: mín$\{v, 1\}$, donde $v = \text{máx}\{(k-j)p_j : l \leq j \leq i\}$
  - El procedimiento de Holland: Ajusta el valor de α en una forma *step-down*, como el método de Holm. Rechaza H_1 a H_{i-1} si i es el menor entero tal que $p_i > 1-(1-\alpha)^{k-i}$
    Holland APV_i: mín$\{v, 1\}$, donde $v = \text{máx}\{1-(1-p_j)^{k-j} : l \leq j \leq i\}$
  - El procedimiento de Finner: Ajusta el valor de α en una forma *step-down*, como el método de Holm y el método de Holland. Rechaza H_1 a H_{i-1} si i es el menor entero tal que $p_i > 1-(1-\alpha)^{(k-1)/i}$
    Finner APV_i: mín$\{v, 1\}$, donde $v = \text{máx}\{1-(1-p_j)^{(k-1)/j} : l \leq j \leq i\}$
- ***Step-up***
  - El procedimiento de Hochberg: Ajusta el valor de α de una forma *step-up*. Funciona comparando el mayor p-valor con α, el siguiente mayor p-valor con $\alpha/2$, y así sucesivamente, hasta que se encuentre una hipótesis que se pueda rechazar. Todas las hipótesis con menor p-valor son rechazadas también.
    Hochberg APV_i: máx$\{(k-j)p_j : (k-1) \geq j \geq i\}$
  - El procedimiento de Hommel: Este es un procedimiento más complicado que el resto, funciona encontrando el mayor j para el que $p_{n-j+k} > k\alpha/j$ para todo $k = 1, ..., j$. Si no existe tal j, se rechazan todas las hipótesis; en otro caso, se rechazan todas que cumplan $p_i \leq \alpha/j$.
    Hommel APV_i: Compruebése el algoritmo para el APV de Hommel en el Algoritmo 12
  - El procedimiento de Rom: Es una modificación del procedimiento de Hochberg para aumentar su potencia. Se comporta de la misma forma que el procedimiento de Hochberg, excepto que los α-valores se calculan a través de la expresión:

$$\alpha_{k-i} = \left[\sum_{j=1}^{i-1} \alpha^j - \sum_{j=1}^{i-2} \binom{i}{k} \alpha_{k-1-j}^{i-j} \right] / i \tag{6.14}$$

    donde $\alpha_{k-1} = \alpha$ y $\alpha_{k-2} = \alpha/2$
    Rom APV_i : máx$\{(r_{k-j})p_j : (k-1) \geq j \geq i\}$, donde r_{k-j} pueden obtenidos a partir de la ecuación 6.14
- **Dos pasos**
  - El procedimiento de Li: Se propuse un procedimiento de rechazo en dos pasos:
    1. Rechazar todas las H_i si $p_{k-1} \leq \alpha$. En otro caso, aceptar las hipótesis asociadas a p_{k-1} e ir al siguiente paso.
    2. Rechazar las H_i restantes con $p_i \leq (1-p_{k-1})/(1-\alpha)\alpha$.
       Li APV_i: $p_i/(p_i + 1 - p_{k-1})$

Algorithm 12 Método para calcular el APV del test de Hommel

procedure (HommelAPV)
  Establecer $APV_i = p_i \ \forall i$
  for each $j = k-1, k-2, ..., 2$ (*en dicho orden*) **do**
    Establecer $B = \emptyset$
    for each $i, i > (k-1-j)$ **do**
      Calcular el valor $c_i = (j \cdot p_i)/(j+i-k+1)$
      Establecer $B = B \bigcup c_i$
    end for
    Encontrar el menor valor c_i en $B \rightarrow c_{min}$
    if $APV_i < c_{min}$ **then**
      Establecer $APV_i = c_{min}$
    end if
    for each $i, i \leq (k-1-j)$ **do**
      Establecer $c_i = \text{mín}(c_{min}, j \cdot p_i)$
      if $APV_i < c_i$ **then**
        Establecer $APV_i = c_i$
      end if
    end for
  end for
end procedure

6.2.4. Estimación de contraste

La estimación de contrate (*Contrast Estimation*) basada en medias puede ser usada para estimar la diferencia entre rendimiento de dos algoritmos. Asume que las diferencias esperadas entre rendimiento de los algoritmos son iguales a través de los problemas. Por lo tanto, el rendimiento de los algoritmos está reflejado por las magnitudes de las diferencias entre ellos en cada dominio.

El interés de este test se basa en la estimación de contrate entre las medias de las muestras de resultados considerando todas las comparaciones por parejas. El test obtiene una diferencia cuantitativa calculada usando las medias entre los dos algoritmos en múltiples problemas, procediendo como sigue:

1. Para cada par de k algoritmos en el experimento, calculamos la diferencia entre el rendimiento de los dos algoritmos en cada uno de los n problemas. Esto es, calculamos las diferencias

$$D_{i(u,v)} = x_{iu} - x_{iv} \tag{6.15}$$

   donde $i = 1, ..., n$; $u = 1, ..., k$; $v = 1, ..., k$.

2. Encontrar la media para cada conjunto de diferencias (Z_{uv}, que puede ser considerado como un *estimador no ajustado* de las medias del algoritmo u y v, $M_u - M_v$). En tanto que $Z_{uv} = Z_{vu}$, solo se necesita calcular Z_{uv} en los casos donde $u < v$. También, nótese que $Z_{uu} = 0$.

3. Calcular la media de cada conjunto de medias no ajustadas teniendo el mismo primer subíndice, m_u:

$$m_u = \frac{\sum_{j=1}^{k} Z_{uj}}{k}, \quad u = 1, ..., k \tag{6.16}$$

4. El estimador de $M_u - M_v$ es $m_u - m_v$, donde $u, v \in [1, k]$.

Estos estimadores se pueden entender como una medida de rendimiento global avanzada. Aunque este test no puede aportar una probabilidad de error asociada con el rechazo de la hipótesis nula de igualdad, es especialmente útil para estimar en cuánta cantidad un algoritmo tiene un mejor rendimiento que otro.

Capítulo 7: Descripción del problema

En este capítulo se presenta el problema que se aborda con el algoritmo creado, así como las necesidades que surgen por el interés de identificar el aporte de este frente al panorama actual del campo en términos de rendimiento, las soluciones que existen para satisfacer dicha necesidad y cuál de ellas es la más adecuada.

7.1. Introducción

El problema que se va a abordar en este proyecto es el de la optimización de problemas de tipo combinatorio.

Definición 7.1 *Un **problema de optimización combinatoria** se define como aquel en el que el conjunto de soluciones posibles es discreto. Es decir, se trata de un problema de optimización que involucra una cantidad finita o numerable de soluciones posibles.*

Este tipo de problemas se diferencia de los problemas de optimización continuos, en los cuales el conjunto de soluciones posibles es infinito e incontable.

Dentro del campo de la optimización combinatoria es común que la mayoría de los procesos de resolución de problemas no puedan garantizar la solución óptima, incluso dentro del contexto del modelo que se esté utilizando. Sin embargo, la aproximación al óptimo suele ser suficiente para resolver los problemas en la práctica.

Con el fin de poder estudiar más a fondo el comportamiento de los distintos algoritmos que se han estado desarrollando era necesario elegir un problema determinado con el que trabajar. En nuestro caso, se ha elegido la generalización del problema comúnmente llamado "problema de la mochila" (*Knapsack Problem* (KP)): ***Quadratic Knapsack Problem* (QKP)**.

Antes de definir el problema, justificaremos por qué se ha elegido este problema. La primera razón, y posiblemente la más importante, es la ausencia de *benchmarks* para problemas combinatorios *expensive*, por lo que nos hemos visto obligados a crear el nuestro propio para un futuro. Ante este panorama también debemos encontrar un problema específico adecuado sobre el que trabajar desde cero y que resulte de interés. Como el objetivo inicial estaba relacionado con redes neuronales, buscamos un problema que puede tener representación binaria (`1` para la elección de elementos, `0` para el caso contrario). En este sentido, el QKP cumple con este requisito, además tiene interés añadido debido a que es un problema con restricciones y constituye una alternativa moderna de un problema clásico; además de que podemos generar instancias de este problema con distintos tamaños. Además, es un problema que tiene muchas aplicaciones en el mundo real en situaciones donde los recursos con distintos niveles de interacción tienen que distribuirse entre distintas tareas, por ejemplo, asignar compañeros de equipo a distintos proyectos donde las contribuciones de cada

miembro se consideran de forma individual y por parejas. Por lo tanto, teniendo en cuenta la falta de *benchmarks* y referencias, el QKP resulta ser una buena opción, ya que es un problema difícil, costoso y moderno.

7.2. Quadratic Knapsack Problem

Se procede a definir en profundidad dicho problema. En primer lugar, se tienen n elementos donde cada elemento j tiene un peso entero positivo w_j. Adicionalmente, se nos da una matriz de enteros no negativos de tamaño $n \times n$, $P = \{p_{ij}\}$, donde p_{jj} es el beneficio asociado a elegir el elemento j y $p_{ij} + p_{ji}$ es el beneficio que se alcanza si ambos elementos i, j, con $i < j$ son seleccionados. Consideramos que una combinación de elementos es una solución a QKP cuando peso el total (la suma del peso de todos los elementos seleccionados) no superan la capacidad máxima de la mochila dada, c. Así, el problema consiste el maximizar el beneficio total sin sobrepasar la capacidad máxima.

Por conveniencia en la notación, establecemos que $N = \{1, \ldots, n\}$ denotará el conjunto de elementos. Representando la lista de elementos de forma binaria, x_j, para indicar si el elemento j ha sido seleccionado (su valor será 0 si no ha sido seleccionado, 1 en caso contrario), el problema podrá ser formulado de la siguiente forma:

$$\begin{aligned} \text{maximizar} \sum_{i \in N} \sum_{j \in N} p_{ij} x_i x_j \\ \text{sujeto a} \sum_{j \in N} w_j x_j \leq c \\ x_j \in \{0, 1\}, j \in N \end{aligned} \tag{7.1}$$

Sin pérdida de generalidad, podemos suponer que:

- $\max_{j \in N} w_j \leq c < \sum_{j \in N} w_j$
- La matriz de beneficios es simétrica, es decir, $p_{ij} = p_{ji}, \forall j > i$.

Una vez definido el problema, es fácil ver por qué es una versión generalizada del KP. KP se puede obtener a partir de QKP si $p_{ij} = 0$, para todo $i \neq j$. También se considera una versión restringida del *Quadratic 0-1 Programming Problem* (QP), el cual se define como 7.1 sin la restricción de capacidad.

Como uno cabría esperar, debido a su generalidad, el QKP tiene un amplio espectro de aplicaciones. Witzgall [32] presentó un problema que surge en telecomunicación cuando un número de localizaciones para satélites tienen que ser seleccionados, tales que el tráfico global entre estas estaciones se maximice y la limitación de presupuesto se cumpla; este problema resulta ser un QKP.

En tanto que QKP es un problema $\mathcal{NP}-hard$, no podemos esperar encontrar una aproximación totalmente polinómica a no ser que $\mathcal{NP} = \mathcal{P}$. Sin embargo, Rader y Woeginger [39] desarrollaron un esquema de aproximación en tiempo completamente polinómico (FPTAS, *Fully polynomial-time approximation scheme*) para el caso especial donde todos los beneficios $p_{ij} \geq 0$. También probaron que si el problema tiene coeficientes de la función objetivo positivos y negativos, entonces el problema es fuertemente $\mathcal{NP} - hard$, por lo que no podemos esperar encontrar ningún FPTAS (a no ser que $\mathcal{NP} = \mathcal{P}$).

7.3. Datos del problema

Utilizaremos 97 archivos de datos generados aleatoriamente de (QKP) *instances*[1], los cuales se pueden distribuir de forma que se indica en la Tabla 7.1.

Tabla 7.1: Datos del Problema

Número de variables	Densidad	Número de archivos
n = 100	25 %	10
	50 %	10
	75 %	10
	100 %	9
n = 200	25 %	9
	50 %	10
	75 %	10
	100 %	10
n = 300	25 %	9
	50 %	10

Se entiende como "densidad" al porcentaje de beneficios combinados positivos, es decir, $p_{ij} > 0$. Particularmente, QP tendría densidad 0 %.

Ahora bien, todos los archivos tienen el mismo formato, lo cual resulta útil para definir funciones capaces de obtener los datos más relevantes para nuestros algoritmos. En primer lugar, mencionar que todos los archivos de datos tienen el mismo formato de nombre:

$$\text{jeu\_}n\text{\_}d\text{\_}x\text{.txt}$$

donde n representa el número de variables que contiene, d la densidad y x el número asociado a dicha instancia.

Dentro de cada archivo de datos se sigue el siguiente formato para representar los datos:

- La referencia de la instancia: Su nombre.
- El número de variables (n)
- Los coeficientes lineales de la función objetivo p_{jj}
- Los coeficientes cuadráticos p_{ij}: representados en líneas
- Una línea en blanco
- 0 si la restricción es de tipo $\leq$, lo cual siempre va a ocurrir ya que estamos considerando instancias QKP.
- La capacidad c de la mochila.
- Los coeficientes de capacidad/peso, w_j.
- Algunos comentarios.

[1] http://cedric.cnam.fr/ soutif/QKP/QKP.html

Un ejemplo breve con 10 variables de qué contendría un archivo y cómo analizarlo sería el siguiente:

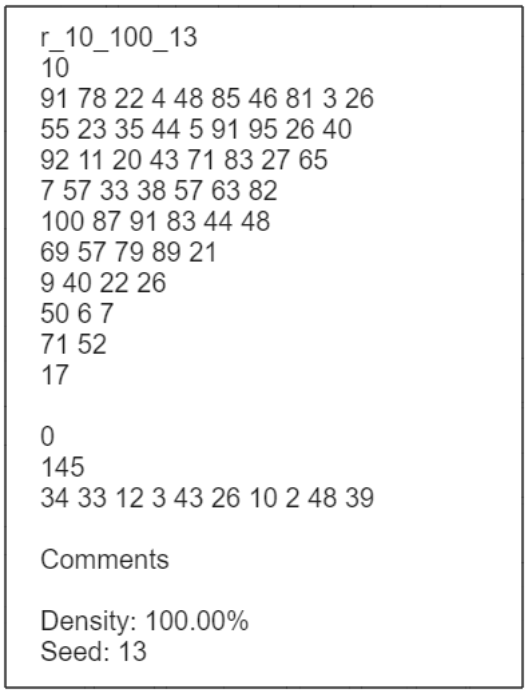


Figura 7.1: Ejemplo de instancia de problema

En este caso se tiene:

- La referencia de la instancia (r_10_100_13)
- El número de variables: 10
- Los coeficientes lineales de la función objetivo p_{ii}: 91 78 22 4 48 85 46 81 3 26
- Los coeficientes cuadráticos de la función objetivo $p_(ij)$: la matriz triangular
- Tras la línea en blanco tenemos un 0 que representa el tipo de restricción (todas las instancias con las que trabajamos en este caso tienen este tipo de restricción).
- La capacidad total: 145.
- La capacidad que necesita cada variable w_i: 34 33 12 3 43 26 10 2 48 39
- Comentarios:
  - La densidad: 100.00 %. Que la podemos obtener de la propia referencia, por lo que no nos es de mucha utilidad este dato.
  - La semilla: 13. Como se explicará más adelante, utilizaremos varias semillas con el fin de que los resultados sean reproducibles y poder utilizar tests estadísticos. Por lo tanto, tampoco nos es de mucha utilidad este dato.

Parte II

Parte informática

Capítulo 8: Algoritmos de Referencia

En este capítulo se presentan los dos algoritmos que se utilizaran como base para desarrollar un algoritmo competitivo para problemas *expensive*, se describirán con sus referencias y se indicarán algunas características; aunque se explicarán de forma más detallada los componentes propios de cada uno en el siguiente capítulo.

8.1. Algoritmo Genético Estacionario Uniforme

En primer lugar, debemos explicar brevemente la importancia de los Algoritmos Genéticos (AG) y, posteriormente, justificar la elección de su versión Algoritmo Genético Estacionario (AGE) (*Steady-State Genetic Algorithm*, SSGA).

Los seres vivos son solucionadores de problemas de forma natural. Exhiben una versatilidad que ponen en evidencia hasta a los mejores programas. Los investigadores más pragmáticos observan el notable poder de la evolución como algo que simular. La selección natural elimina uno de los mayores inconvenientes en el diseño de software: especificar de antemano todas las características de un problema y las acciones que dicho programa tendría que tomar para tratar con ellas. Aprovechando los mecanismos de evolución, los investigadores pueden ser capaces de "reproducir" programas que ayuden con la resolución de problemas cada vez más complejos. Efectivamente, estos llamados **algoritmos genéticos** han demostrado la habilidad de hacer avances en el diseño de sistemas complejos.

Los AGs hacen posible explorar un rango mucho más amplio de posibles soluciones a un problema que programas convencionales. Además, en los estudios realizados sobre la selección natural de programas bajo condiciones controladas bien entendidas, los resultados prácticos alcanzados pueden aportar cierto conocimiento sobre los detalles de cómo la vida y la inteligencia evolucionan en el mundo natural.

El funcionamiento de un AG viene dado por el siguiente pseudocódigo (Algoritmo 13).

En el caso de la versión AGE, su pseudocódigo viene representado en 14. Claramente sigue la misma estructura, pero presenta una especificación:

- Estacionario (E): En relación con el operador de selección. Se enfrentan las soluciones hijas con las de la población de la generación anterior y mantenemos las mejores.

También se estuvieron barajando otra versión estudiada durante la asignatura de Metaheurísticas:

- Generacional (G): En relación con el operador de selección. Sustituimos la antigua generación por la nueva.

Algorithm 13 Algoritmo Genético

procedure (AG)($EMax > 0, nelem > 0, pcruce \in [0,1], pmut \in [0,1]$)
    Generar una población inicial aleatoria
    Calcular la función *fitness* de cada individuo
    `generacion` $= 0$
    while `generacion` <EMax **do**
        Calcular el número de parejas a formar para el cruce $\rightarrow ncruce = pcruce * nelem$
        Seleccionar aleatoriamente con repetición $4 * ncruce$ soluciones de la población
        Aplicar torneo de 2 en 2 soluciones del conjunto anterior y almacenar solo la mejor $\rightarrow$ `padres`
        for $i \in [0, ncruce]; i+ = 2$ **do**
            Generar 2 hijos cruzando `padres`$[i]$ y `padres`$[i+1]$
            Aplicar el Operador de Reparación sobre ambos hijos
            Calcular la función *fitness* de cada hijo
            Almacenar dichos hijos $\rightarrow$ `hijos`
        end for
        Calcular el número de soluciones a mutar $\rightarrow nmut = pmut * nelem$
        Mutar $nmut$ soluciones distintas
        Calcular la función *fitness* de las nuevas soluciones
        Aplicar el Operador de Selección sobre la población actual e `hijos`
        `generacion` = `generacion`+1
    end while
end procedure

La versión generacional puede ocasionar que se pierda la mejor solución hasta el momento, lo que impediría que se pudiese seguir una búsqueda profundizando en dicha solución. Como tenemos pocas iteraciones, no nos podemos permitir buenas soluciones sin ningún tipo de garantía, así que no sería un buen enfoque inicial.

A efectos prácticos, se ha usado los resultados y el análisis que realicé en el trabajo "Problemas con técnicas basadas en poblaciones" de la asignatura Metaheurísticas (curso 2021-2022), donde se debía comparar experimental y teóricamente los distintos algoritmos genéticos y meméticos. En dicho trabajo se llega a la conclusión de que la mejor opción es utilizar AGE. En tanto que no es el mismo problema, no podemos garantizar que este sea el caso para este problema con solo esa información, pero sabemos que es un buen punto de partida.

8.1.1. Pseudocódigo

A efectos prácticos, este pseudocódigo (Algoritmo 14) se diferenciará del anterior (Algoritmo 13) en el cruce, ya que solo cruzaremos dos padres no necesitaremos el parámetro *pcruce*, y que en el operador de reemplazo concretaremos que es estacionario.

Como se ha dicho al principio de este capítulo, una explicación más detallada junto con el pseudocódigo de cada una de las componentes será dada en el siguiente capítulo.

Algorithm 14 Algoritmo Genético Estacionario

```
1: procedure (AG)(EMax > 0, nelem > 0, pmut ∈ [0, 1])
2:     Generar una población inicial aleatoria
3:     Calcular la función fitness de cada individuo
4:     generacion = 0
5:     while generacion <EMax do
6:         Seleccionar aleatoriamente 4 soluciones de la población sin repetición 2 a 2
7:         Aplicar torneo de 2 en 2 soluciones del conjunto anterior y almacenar solo la mejor → padres
8:         Generar 2 hijos cruzando padres_1 y padres_2
9:         Aplicar el Operador de Reparación sobre ambos hijos
10:        Calcular la función fitness de cada hijo
11:        Almacenar dichos hijos → hijos
12:        Calcular el número de soluciones a mutar → nmut = pmut * nelem
13:        Mutar nmut soluciones distintas
14:        Calcular la función fitness de las nuevas soluciones
15:        Aplicar el Operador de Selección Estacionario sobre la población actual e hijos
16:        generacion = generacion+1
17:    end while
18: end procedure
```

8.1.2. Componentes

Operador de Reparación

Cuando se realiza el cruce de dos soluciones pueden ocurrir dos casos:

- El resultado del cruce sea una solución factible.
- El resultado del cruce no sea una solución factible.

Más adelante en este capítulo se explicarán cómo son los cruces y se entenderá por qué es posible que el resultado de un cruce sea una solución no factible. En este caso, no es lógico desechar al "hijo", por lo que debemos "arreglarlo" para que se vuelva una solución. Ese es el objetivo del Operador de Reparación.

En nuestro caso lo vamos a aplicar en ambos casos (que el "hijo" sea solución o no). Si el "hijo" no es solución factible es obvio el por qué necesitamos aplicarlo. Si el "hijo" es solución factible lo aplicaremos para asegurarnos que no hay huecos libres, es decir, que no hay elementos adicionales que podrían introducirse adicionalmente. Esto último se hace ya que queremos maximizar el valor de la solución, por lo que si queremos que sea mínimamente competente para cuando intentemos introducirlas en la población.

Entonces seguimos el siguiente proceso, ilustrado en el pseudocódigo 15:

- Si no es solución factible (su peso supera al peso máximo): Se deberán eliminar elementos del "hijo" hasta que constituya una solución factible. La forma de eliminar elementos será usando el algoritmo Greedy, es decir, se eliminarán los elementos con menor proporción *valor_acumulado/peso*. Esta lógica viene dada por un intento de eliminar el máximo peso posible sin reducir mucho el valor final cuando se vuelva una solución.

- Si es solución factible (su peso no supera al peso máximo): Se buscará, utilizando el algoritmo Greedy, un elemento para introducir. En este caso, nos interesa encontrar el elemento con mayor proporción *valor_acumulado/peso*, ya que eso nos permitiría potencialmente aumentar significativamente el valor total (evaluación de la función *fitness*) de la solución. Esto es, al tener en cuenta el peso puede llegar a resultar en que seamos capaces de introducir más elementos, pudiendo superar finalmente el valor que se tendría si se introdujese el elemento con el mayor valor acumulado pero no permitiese introducir más elementos. Este proceso se repetirá hasta que no sea capaz de introducir ningún otro elemento en la solución.

Algorithm 15 Operador de Reparación

```
1:  procedure (Operador Reparación)(hijo)
2:      Calcular el peso total de hijo → pesoHijo
3:      if pesoHijo >c then
4:          while pesoHijo >c do
5:              Eliminar elemento usando Greedy
6:          end while
7:      else
8:          anadido = true
9:          while anadido do
10:             anadido = Añadir elemento usando Greedy
11:         end while
12:     end if
13: end procedure
```

Téngase en cuenta que al eliminar y añadir elementos del "hijo", se debe recalcular su peso total.

Operador de Cruce

El cruce es un operador genético usado para crear nuevas soluciones a partir de otras soluciones que podrán llegar a formar parte de la población de la próxima generación. Dos soluciones obtenidas de la población actual se cruzarán con el objetivo de producir una descendencia que sea parecida a los padres, para heredar sus buenas características; resultando en una solución superior.

Nos encontramos con distintos tipos de cruces básicos:

- **Cruce en un punto**: Dados dos padres, se le asignan los elementos de $\texttt{padre}_1$ a $\texttt{hijo}_1$ y de $\texttt{padre}_2$ a $\texttt{hijo}_2$ hasta cierto cromosoma elegido con anterioridad. A partir de dicho cromosoma, cambiaremos la asignación de forma que $\texttt{hijo}_1$ hereda de $\texttt{padre}_2$ e $\texttt{hijo}_2$ hereda de $\texttt{padre}_1$. Un ejemplo de este tipo de cruce podría ser:

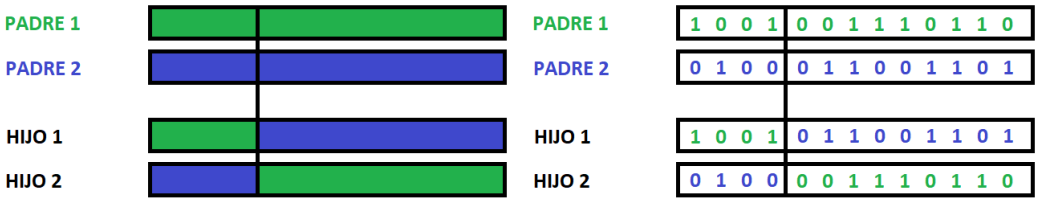


Figura 8.1: Cruce en un punto

- **Cruce en dos puntos**: Sigue la misma lógica que el anterior, solo que elegimos dos puntos a partir de los cuales se cambian los elementos de qué padre se asignan a cada hijo. Un ejemplo de este tipo de cruce podría ser:

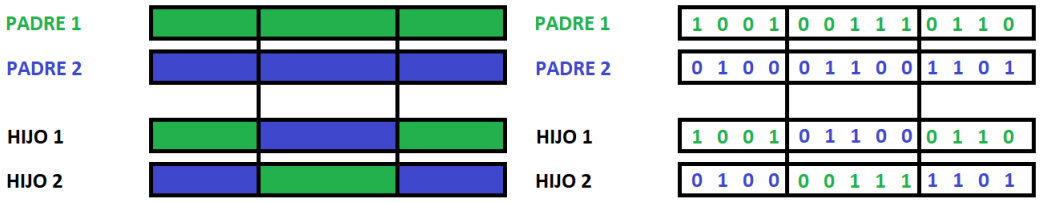


Figura 8.2: Cruce en dos puntos

- **Cruce uniforme**: En este caso, en cada cromosoma se elige de forma aleatoria de qué padre lo hereda, cumpliéndose que si un hijo hereda cierto cromosoma de un padre, el otro hijo deberá heredar el mismo cromosoma del otro padre. Un ejemplo de este tipo de cruce sería:

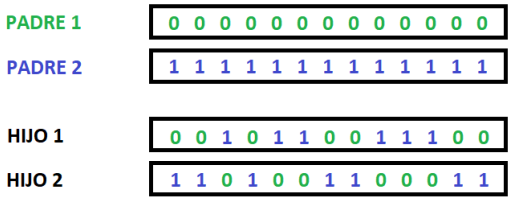


Figura 8.3: Cruce Uniforme

El cruce de dos soluciones buenas no tiene por qué siempre dar lugar a una solución mejor o igual de buena. Sin embargo, si los padres son buenas soluciones, la probabilidad de tener un hijo bueno es elevada; en el caso de que el hijo no sea una buena solución, será eliminado durante el periodo de reemplazo.

En nuestro problema es posible que el cruce de dos soluciones no de lugar a una solución factible. Esto se debe a la elección aleatoria de qué cromosomas elegir, no estamos teniendo en cuenta el peso que se está alcanzando al asignar cada elemento; por lo que es totalmente posible que al asignar los elementos a cada hijo se sobrepase la capacidad máxima, dejando por ello de ser una solución factible.

En nuestro caso, realmente utilizamos una mezcla de cruce en un punto y cruce uniforme. Esto es, vamos a asignarle a cada hijo la mitad de cada uno de los padres, pero esta asignación será aleatoria: desordenamos el orden de los índices y lo partimos por la mitad. Esto viene representado en el pseudocódigo 16.

Mutación

Los primeros intentos de mezclar computación y evolución no progresaron porque pusieron énfasis en los textos de biología del momento y confiaban más en la mutación que en el cruce para generar nuevas combinaciones de genes.

La mutación consiste en modificar al azar una muy pequeña parte del cromosoma de los individuos, y permite alcanzar zonas del espacio de búsqueda que no estaban cubiertas por los individuos

Algorithm 16 Cruce Uniforme

```
1:  procedure (Cruce Uniforme)(padre_1, padre_2)
2:      Desordenar los índices que indican la posición de cada elemento
3:      for i in 0..n do
4:          if i <n/2 then
5:              hijo_1[indice[i]] = padre_1[indice[i]]
6:              hijo_2[indice[i]] = padre_2[indice[i]]
7:          else
8:              hijo_1[indice[i]] = padre_2[indice[i]]
9:              hijo_2[indice[i]] = padre_1[indice[i]]
10:         end if
11:     end for
12: end procedure
```

de la población actual. La mutación sola de por sí generalmente no permite avanzar en la búsqueda de una solución, pero nos garantiza que la población no va a evolucionar hacia una población uniforme que no sea capaz de seguir evolucionando.

De forma similar a lo explicado en el Operador de Reparación, una vez que obtengamos la nueva solución debemos comprobar si podemos introducir más elementos, con el fin de maximizar el valor que puede llegar a tener. Esto también lo haremos siguiendo la misma lógica: ir introduciendo los genes con mayor proporción *valor_acumulado/peso*.

El pseudocódigo de nuestra implementación de la mutación viene expresada en Algoritmo 17.

Algorithm 17 Mutación

```
1:  procedure (Mutación)(poblacion, prob_mut)
2:      Calcular el número de cromosomas que mutarán → nmut = n*prob_mut
3:      Almacenar de forma aleatoria sin repetición nmut cromosomas de poblacion → mutacion
4:      for i in 0..nmut do
5:          Elegir dos genes con distinto valor de forma aleatoria
6:          if Al intercambiar los genes sigue siendo válido then
7:              Intercambiar los genes
8:          else
9:              Volver al paso anterior
10:         end if
11:         anadido = true
12:         while anadido do
13:             anadido = Añadir elemento usando Greedy
14:         end while
15:         Calcular el valor de la función fitness
16:     end for
17: end procedure
```

Operador de Reemplazo Estacionario

Una vez que hemos generado nuevas soluciones a partir del cruce de soluciones de la población existente, es necesario establecer un criterio sobre qué soluciones se mantienen o insertan en la

población para la siguiente generación. En esta versión del Operador de Reemplazo el criterio que se sigue es el enfrentamiento de al población actual con las soluciones hijas, la población resultante estará compuesta de aquellos con mayor valor al calcular su función *fitness*.

En concreto, para lo que se aplica en el algoritmo base, AG, tenemos en cuenta que solo generamos 2 soluciones hijas antes de enfrentarlas con la población. Por ello, se ha optado por un método simplificado de enfrentamiento que consiste en encontrar cuáles son las 2 peores soluciones de la población actual (es decir, aquellas soluciones de la población actual con menor valor *fitness*) y comprobar si las hijas son mejores o no. Se puede seguir su esquema en el pseudocódigo 18. Usaremos los índices de forma que indique un orden de valores *fitness*, por ejemplo, dados $\texttt{padre}_1$ y $\texttt{padre}_2$, se cumplirá que *fitness*($\texttt{padre}_1$) >*fitness*($\texttt{padre}_2$). Además, por conveniencia en la notación, se entenderá que $\texttt{solucion}_i$ >$\texttt{solucion}_j$ significa que el valor *fitness* de $\texttt{solucion}_i$ es mayor al de $\texttt{solucion}_j$.

Más adelante en la sección de "Componentes de CHC" de este capítulo se presenta la versión más generalizada de este tipo de enfrentamiento.

Algorithm 18 Operador de Reemplazo Estacionario

```
1: procedure (Op Reemplazo Estacionario)
2:     Calcular los 2 peores padres de la población actual → padre_1, padre_2.
3:     if hijo_1 >padre_1 && hijo_2 >padre_2 then
4:         Intercambiar ambas soluciones de la población por ambos hijos
5:     else if hijo_1 >padre_2 then
6:         Intercambiar la peor solución de la población por el mejor hijo
7:     else
8:         No hacer nada, ya que los dos hijos son peores que las peores soluciones de la población
9:     end if
10: end procedure
```

8.2. CHC

El algoritmo CHC utiliza un método de selección elitista que, combinada con un mecanismo de prevención de incesto y un método para obligar que la población diverja cada vez que converge, permite el mantenimiento de la diversidad de la población. Este algoritmo se ha utilizado de forma exitosa en el pasado para problemas de optimización estáticos.

El algoritmo CHC (*Cross-generational elitist selecition, Heterogeneous recombination and Cataclysmic mutation*) propuesto por Eshelman utiliza un método de selección elitista combinado con un cruce altamente disruptivo para promover la diversidad de la población. La principal característica de este algoritmo es su capacidad de prevenir la convergencia de la población, algo que, como luego comprobaremos, será útil en nuestro problema.

Originalmente, cuando la población converge se pueden tomar dos acciones:

- Reiniciar la población entera de forma aleatoria con excepción de la mejor solución
- Reiniciar la población utilizando la mejor solución como base y generando el resto realizando modificaciones sobre esta.

En nuestro problema realizar esto resultaría en una pérdida de tiempo e iteraciones importantes, por lo tanto, no se tendrá en cuenta.

8.2.1. Pseudocódigo

Algorithm 19 Algoritmo CHC

1: **procedure** (AG)($EMax > 0, nelem > 0$)
2: Generar una población inicial aleatoria
3: Calcular la función *fitness* de cada individuo
4: `generacion` $= 0$
5: `threshold` $= n/4$
6: **while** `generacion` <EMax **do**
7: Calcular el número de parejas a formar para el cruce $\rightarrow ncruce = pcruce * nelem$
8: Desordenar los elementos de la población actual y comprobar si cumplen la condición de prevención de incesto (distancia de Hamming >`threshold`) de dos en dos
9: **if** `hamming` >`threshold` **then**
10: Almacenar las soluciones $\rightarrow$ `parejas`
11: **end if**
12: **if** `parejas` $= \emptyset$ **then**
13: **if** `threshold` $\neq 0$ **then**
14: `threshold` $=$ `threshold`-1
15: **end if**
16: **else**
17: **for** i$\in [0,$ `parejas.size()`$]$; i=i+2 **do**
18: Generar 2 hijos cruzando `parejas`$[i]$ y `parejas`$[i+1]$
19: Aplicar el Operador de Reparación sobre ambos hijos
20: Calcular la función *fitness* de cada hijo
21: Almacenar dichos hijos $\rightarrow$ `hijos`
22: **end for**
23: Aplicar el Operador de Selección sobre la población actual e `hijos`
24: **end if**
25: `generacion` $=$ `generacion`+1
26: **end while**
27: **end procedure**

8.2.2. Componentes

Operador de reparación

Se utilizará el mismo que se ha presentado anteriormente en los componentes de AG. Véase el pseudocódigo 15.

Cruce HUX

El cruce HUX (*Half Uniform Crossover*) se caracteriza por, dados dos cromosomas, asignarle a los resultados del cruce en primer lugar los genes comunes a ambos padres y el resto de los genes serán repartidos a partes iguales entre ambos padres. Es decir, exactamente la mitad de los genes no coincidentes se intercambian en los hijos.

A continuación se detallará en forma de pseudocódigo (20) el comportamiento de este tipo de cruce. Aunque primero se mostrará un ejemplo de este tipo de cruce en Figura 8.4.

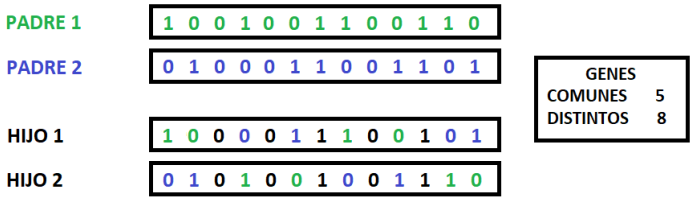


Figura 8.4: Cruce HUX

Algorithm 20 Cruce HUX

procedure (Cruce HUX)($padre_1, padre_2$)
    Asignar los genes comunes de los padres a ambos hijos
    Desordenar los índices que indican la posición de cada gen restante → Supongamos tamaño $m \leq n$
    for i in $0..m$ **do**
        if i $<m/2$ **then**
            $\texttt{hijo}_1\texttt{[indice[i]]} = \texttt{padre}_1\texttt{[indice[i]]}$
            $\texttt{hijo}_2\texttt{[indice[i]]} = \texttt{padre}_2\texttt{[indice[i]]}$
        else
            $\texttt{hijo}_1\texttt{[indice[i]]} = \texttt{padre}_2\texttt{[indice[i]]}$
            $\texttt{hijo}_2\texttt{[indice[i]]} = \texttt{padre}_1\texttt{[indice[i]]}$
        end if
    end for
end procedure

Reemplazo Elitista

Es una generalización del Operador de Reemplazo Estacionario del AG (Algoritmo 18). En este caso tenemos que enfrentar toda la población de hijos (con tamaño menor igual al tamaño de la población) contra toda la generación anterior.

Para este caso se ha optado por, a grandes rasgos, juntar ambas poblaciones (generación actual y descendientes) en una sola y ordenarlas en base a su valor *fitness*. De esta forma, al final tenemos todas nuestras soluciones ordenadas de mejor a peor y solo tenemos que quedarnos con las n para formar la nueva población.

Este comportamiento se puede ver reflejado en su pseudocódigo (Algoritmo 21)

Algorithm 21 Enfrentamiento CHC

procedure (Enfrentamiento)($poblacion, hijos$)
    Calcular la función *fitness* de `poblacion` e `hijos`
    Unir ambas poblaciones → `pobTotal`
    Unir los valores de ambas poblaciones → `valorTotal`
    Ordenar los elementos de `pobTotal` según `valorTotal` (orden descendente)
    Asignar a la población actual los n primeros elementos de `pobTotal`
end procedure

Capítulo 9: Componentes de la propuesta

En este capítulo se presentan los distintos componentes que se han ido desarrollando como modificaciones de los distintos componentes de los algoritmos de referencia. Es importante mencionar que en este capítulo no solo se harán mención de aquellas modificaciones que han resultado fructuosas, si no también aquellas que no han mejorado los resultados de versiones anteriores. Estas últimas se conformarán su propia sección aparte al final del capítulo. Se describirán detalladamente además de usar pseudocódigo para representarlos.

Téngase en cuenta que solo se explicará brevemente el por qué de estas modificaciones, ya que entraremos en este tema en más profundidad en el siguiente capítulo.

9.1. Histórico

Al tener tan pocas evaluaciones y con un tamaño tan pequeño de la población no podemos permitirnos una exploración del vecindario de las distintas soluciones en direcciones que en el momento se podrían considerar erróneas, es decir, que ocasionarían que empeorasen las soluciones. Por ello, se plantea un sistema capaz de "recordar" buenos elementos y malos elementos de la solución; considerándose "buenos elementos" aquellos que aparecen en gran medida en las mejores soluciones y no se encuentran en las peores, y "malos elementos" aquellos que aparecen en gran medida en las peores soluciones y no se encuentran en las mejores. A esto lo llamaremos **histórico**.

De esta forma, podemos guiar la exploración de vecindario hacia aquellos elementos que han demostrado ser "buenos" y alejarlos de aquellos que han demostrado ser "malos". Esta lógica la podemos aplicar en el Operador de Reparación y/o en la Mutación. Esto se haría sustituyendo la lógica Greedy de tener en cuenta el ratio *valor_acumulado/peso* a la hora de añadir o eliminar elementos por un fomento del uso de los datos del histórico: se intentarán añadir de forma aleatoria entre los mejores elementos y se intentarán eliminar de forma aleatoria entre los peores elementos.

Sin embargo, no podemos permitir que esto guíe toda la ejecución, tenemos que decidir un par de cosas:

- ¿Cuándo se para de recopilar datos para generar el histórico? Si recopilamos información de pocas iteraciones puede ocasionar que los datos obtenidos no sean representativos, ya que es posible que no se hayan alcanzado en el momento suficientes soluciones buenas. Si recopilamos información de demasiadas iteraciones puede suponer que los datos obtenidos no sean representativos, ya que la población hubiese podido converger por lo que no hay ningún elemento que se pudiese considerar "bueno" o "malo"; además de que si se dejan pocas ejecuciones para el uso del histórico no se va a poder alcanzar mucha mejora.
- ¿Durante cuántas ejecuciones deberíamos utilizar el histórico? En otras palabras, ¿una vez obtenido los datos del histórico deberíamos utilizarlos hasta el final del algoritmo? La res-

puesta a esto es: no se debería. Esto es porque no podemos esperar que los datos obtenidos mediante el histórico sean del todo fiables, es decir, podrían estar guiando la solución hasta un máximo local, por lo que las soluciones de la población convergerían rápidamente.

Por lo tanto, llegamos a la conclusión de que no podemos utilizar únicamente el histórico. Por ello, lo que haremos será intercalar varias etapas de recopilación de datos y uso del histórico generado. Durante la recopilación de datos se procederá de la misma forma que lo haría si esta modificación no estuviese añadida, con la excepción de que se irá guardando en un archivo las distintas soluciones que estamos alcanzando para luego hacer el estudio de sus elementos. Cada etapa tiene una duración de 50 iteraciones seguidas, con lo que se tendría que cada uso individual del histórico estaría compuesto de 100 iteraciones (50 iteraciones de recopilación de datos y 50 iteraciones de uso de dichos datos). De esta forma, repetimos este proceso 4 veces y las 50 iteraciones restantes se dedicarán al comportamiento usual del algoritmo (por simplicidad a la hora de implementar el código, esto es equivalente a la repetición de una etapa de recopilación de datos). Este esquema se ve representado de forma genérica, por simplicidad, en el Algoritmo 22.

Algorithm 22 Estructura General del Histórico

```
1:  procedure (Histórico)
2:      RecDatos = True
3:      for i = 0; i <NEVALUACIONES; ++i do
4:          if i%50==0 && i≠ 0 then /* Puntos de cambio */
5:              if i%100 then /* Empezamos a recopilar datos */
6:                  RecDatos = True
7:                  Eliminar todos los datos actuales del histograma
8:                  Recopilar las soluciones de la población actual
9:              else
10:                 RecDatos = False
11:             end if
12:         end if
13:         if RecDatos then
14:             Ejecutar los componentes de la manera original
15:             Recopilar las nuevas soluciones
16:         else
17:             Ejecutar los componentes usando los datos del histórico
18:         end if
19:     end for
20: end procedure
```

Las modificaciones realizadas al Operador de Reparación y a la Mutación se verán representadas respectivamente en Algoritmo 23 y Algoritmo 24. Para el pseudocódigo 24 téngase en cuenta la siguiente notación:

- `Mejores`: Aquellos elementos que suelen estar presentes en las mejores soluciones, pero no en las peores.
- `Peores`: Aquellos elementos que suelen estar presentes en las peores soluciones, pero no en las mejores.
- `Sin información`: Aquellos elementos que frecuentemente no aparecen en las mejores ni en

las peores soluciones o que aparecen en ambas (por lo que no nos aporta ninguna información ya que no se puede considerar un elemento determinista en lo buena que es una solución).

Algorithm 23 Histórico en Operador de Reparación

```
1: procedure (Historico_OR)(hijo)
2:     Calcular el peso total de hijo → pesoHijo
3:     if pesoHijo >c then
4:         Eliminar de forma aleatoria los peores elementos según el histórico hasta que pesoHijo
   deje de sobrepasar c
5:         Si no es posible lo anterior y sigue dándose la condición, eliminar de forma aleatoria los
   elementos que no aparecen en el histórico hasta que pesoHijo deje de sobrepasar c
6:         Si no es posible lo anterior y sigue dándose la condición, eliminar de forma aleatoria los
   mejores elementos según el histórico hasta que pesoHijo deje de sobrepasar c
7:     else
8:         Añadir de forma aleatoria los mejores elementos según el histórico sin que pesoHijo
   sobrepase c
9:         Añadir de forma aleatoria los elementos que no aparecen en el histórico sin que pesoHijo
   sobrepase c
10:        Añadir de forma aleatoria los peores elementos según el histórico sin que pesoHijo
   sobrepase c
11:    end if
12: end procedure
```

9.2. GRASP

En el capítulo 6 se ha explicado por qué seguimos un enfoque *Greedy* en el Operador de Reparación. Sin embargo esto tiene un problema, y esto es que es bastante probable que siempre se estén eligiendo los mismos pocos elementos cada vez que se usa este componente. Si realmente ocurre eso, ocasionaría que la población converja y la evolución se estanque; por ello, debemos encargarnos de encontrar otra alternativa con la que podamos aumentar la diversidad de la población.

Sin embargo, la diversidad en la población es un arma de doble filo. La diversidad es necesaria para explorar el espacio de soluciones, pero puede causar que no finalice el proceso de exploración. Si este es el caso, entonces no se le dedicará suficiente tiempo a la fase de explotación, que es esencial para obtener soluciones de mejor calidad. Por lo tanto, utilizaremos un operador capaz de introducir cierta diversidad a la población a la vez que asegura cierta calidad. Esto es, utilizaremos el operador GRASP (*Greedy Randomized Adaptive Search Procedure*) [40].

El GRASP vendrá presentado en Algoritmo 25. De forma simplificada, GRASP utiliza de base el algoritmo Greedy para obtener cierta cantidad de mejores elementos, eligirá aleatoriamente uno de ellos y lo devolverá. Lo que se entiende como "mejor elemento" para GRASP es lo mismo que para lo que se propuso durante la justificación del operador Greedy. Además, la cantidad de elementos a considerar dependerá del valor aportado por el mejor elemento; en concreto, solo se considerarán los elementos cuyo valor relativo (*valor_acumulado/peso*) varíe en un 10 % del valor acumulado del mejor elemento, es decir:

- Si se necesitan añadir elementos, nos quedaremos con los elementos que tengan un valor relativo mayor que $0.9 * valor\_relativo\_mayor$.

Algorithm 24 Histórico en Mutación

```
1: procedure (Historico_Mutacion)(solucion)
2:     sustituido = False
3:     if !sustituido && Se puede sustituir un peor elemento por un mejor elemento then
4:         Sustituir peor por mejor
5:         sustituido ← true
6:     else if !sustituido && Se puede sustituir un sin información elemento por un mejor
   elemento then
7:         Sustituir sin información por mejores
8:         sustituido ← true
9:     else if !sustituido && Se puede sustituir un peor elemento por un sin información
   elemento then
10:        Sustituir peores por sin información
11:        sustituido ← true
12:    else if !sustituido && Se puede sustituir un sin información elemento por un sin
   información elemento then
13:        Sustituir sin información por sin información
14:        sustituido ← true
15:    else if !sustituido && Se puede sustituir un mejores elemento por un mejores elemento
   then
16:        Sustituir mejores por mejores
17:        sustituido ← true
18:    else if !sustituido && Se puede sustituir un peores elemento por un peores elemento
   then
19:        Sustituir peores por peores
20:        sustituido ← true
21:    else if !sustituido && Se puede sustituir un mejores elemento por un sin información
   elemento then
22:        Sustituir mejores por sin información
23:        sustituido ← true
24:    else if !sustituido && Se puede sustituir un sin información elemento por un peores
   elemento then
25:        Sustituir sin información por peores
26:        sustituido ← true
27:    else if !sustituido && Se puede sustituir un mejores elemento por un peores elemento
   then
28:        Sustituir mejores por peores
29:        sustituido ← true
30:    end if
31:    Aplicar Operador de Reparación con Histórico para rellenar más las soluciones
32: end procedure
```

- Si se necesitan eliminar elementos, nos quedaremos con los elementos que tengan un valor relativo menor que $1.1 * valor\_relativo\_menor$.

Algorithm 25 Operador GRASP

```
1: procedure (GRASP)(solucion, min)
2:     if min then
3:         Almacenar en indices los elementos activados en solucion
4:     else
5:         Almacenar en indices los elementos desactivados en solucion
6:     end if
7:     Calcular el valor relativo de indices → valores
8:     encontrado = false
9:     if min then
10:        Ordenar indices según valores en orden ascendente
11:        for i ∈ 1..size(indices)&&!encontrado do
12:            if valores_i >1.1·valores_0 then
13:                encontrado = true
14:                Eliminar los elementos de indices desde i hasta size(indices)
15:            end if
16:        end for
17:    else
18:        Ordenar indices según valores en orden descendente
19:        for i ∈ 1..size(indices)&&!encontrado do
20:            if valores_i <0.9·valores_0 then
21:                encontrado = true
22:                Eliminar los elementos de indices desde i hasta size(indices)
23:            end if
24:        end for
25:    end if
26:    Elegir aleatoriamente algún elemento de indices
27: end procedure
```

9.3. Cruce Intensivo

Para aumentar la explotación de buenas soluciones se propone hacer una modificación en el cruce que realizamos. En vez de utilizar el cruce uniforme, que hemos establecido que a cada hijo se le asigna la mitad de los genes de cada uno de los padres (aunque sea de forma aleatoria), en este caso se le asignará un mayor porcentaje de genes del mejor de los padres.

Para ello, lógicamente tendremos que calcular cuál de los padres es la mejor solución (tiene mayor valor). En este caso, no nos sirve aleatorizar una sola lista de índices, ya que la separación no va a ser igualitaria. Por lo que tendremos que hacer las separaciones de genes para ambos hijos. Una vez hecho esto, se procede de igual forma que en los anteriores cruces, es decir, se le asigna a cada hijo los genes de los padres correspondientes y, posteriormente, se le aplica el operado de reparación a ambos hijos. Estoy viene representado en Algoritmo 26.

Por conveniencia, vamos a volver a utilizar la notación que implica que $\texttt{padre}_1$ tiene un valor más elevado que $\texttt{padre}_2$.

Algorithm 26 Cruce Intensivo

```
1: procedure (Cruce Intensivo)(padre_1, padre_2, porcentaje)
2:     Desordenar los índices (indices_1, indices_2) que indican la posición de cada elemento
3:     nelem = n*porcentaje
4:     for i in 0..n do
5:         if i <nelem then
6:             hijo_1[indice_1[i]] = padre_1[indice_1[i]]
7:             hijo_2[indice_2[i]] = padre_1[indice_2[i]]
8:         else
9:             hijo_1[indice_1[i]] = padre_2[indice_1[i]]
10:            hijo_2[indice_2[i]] = padre_2[indice_2[i]]
11:        end if
12:    end for
13: end procedure
```

9.4. Diversidad en la Población Inicial

Uno de los mayores problemas que se puede tener cuando la población es pequeña y no nos podemos permitir muchas ejecuciones es la falta de diversidad en dicha población. Si todos los elementos son muy parecidos entre sí resultará muy difícil llevar a cabo una correcta exploración del espacio de soluciones, ya que las soluciones generadas en el cruce se seguirán pareciendo a sus padres (por lo que serán similares al resto de la población) y en la mutación, al solo intercambiar un par de elementos no se va a conseguir una solución lo suficientemente distinta del resto.

Por ello, lo primero que debemos garantizar antes de empezar el algoritmo es que tenemos una población inicial lo suficientemente diversa sobre la que trabajar. Esto viene representando en el pseudocódigo 27. Una población inicial diversa se consigue generando las soluciones de forma aleatoria sucesivamente a la vez que se comprueba su similitud (su distancia de Hamming) con respecto al resto de soluciones que ya forman parte de la población. Si dicha nueva solución no supera un umbral de diferencias (normalmente representado como un porcentaje del número de elementos totales del problema) con respecto a todas las soluciones introducidas, se descarta esta solución y se vuelve a generar otra. Este proceso se repite hasta haber alcanzado el tamaño de población necesario para empezar la ejecución del algoritmo.

Donde recordemos que n es el número de variables del problema y `numcro` el tamaño de la población con la que trabajaremos. *p_delta* representa el porcentaje de elementos del número de variables total que deben ser distintos para considerar que la nueva solución es lo suficientemente diferente al resto de soluciones introducidas en la población; por lo que `distMin` será dicho número mínimo de elementos distintos.

9.5. Operador NAM

El operador de Emparejamiento Variado Inverso (*Negative Assortative Mating*) o NAM [28] está orientado a generar diversidad. Esta es una alternativa a cómo se eligen los individuos de la población que se van a cruzar. La lógica de este operador es parecida a la prevención de incesto que se propone en el algoritmo CHC, pero sin ser tan restrictiva. Se necesita que los padres sean distintos para conseguir que los hijos aporten diversidad a la población si llegan a ser introducidos. Una optativa a esto es el operador NAM. En este trabajo se utilizará una versión del NAM.

Algorithm 27 Población Inicial Diversa

```
1: procedure (Población Diversa)(p_δ)
2:     distMin ← n · p_δ
3:     npob ← 0
4:     poblacion ← ∅
5:     while npob<numcro do
6:         Generamos una solución aleatoria → solucion_npob
7:         nuevo ← true
8:         for each sol∈poblacion do
9:             if distHamming(solucion_npob,sol) <distMin then
10:                nuevo ← false
11:            end if
12:        end for
13:        if nuevo then
14:            Añadir solucion_npob a poblacion
15:            npob++
16:        end if
17:    end while
18: end procedure
```

La implementación de este operador viene representado en el Algoritmo 28. Lo que se hace en este operador es elegir a una solución utilizando el Torneo Binario propio de los algoritmos genéticos (en el operador original simplemente se elige un individuo de la población de forma aleatoria). Para la segunda solución, primero, debemos obtener algunas soluciones de la población distintas a la primera obtenida para el cruce. Una vez hecho esto, elegimos como el segundo padre a la solución del segundo grupo más diferente al primer padre; esto es, se calculará la distancia de Hamming entre el primer padre y las soluciones del segundo grupo, y se elegirá como segundo padre aquella solución con mayor distancia de Hamming.

Algorithm 28 Operador NAM

```
1: procedure (Operador NAM)
2:     Aplicar Torneo Binario para obtener al primer padre → padre_1
3:     Elegir aleatoriamente sin repetición un subconjunto de la población distinta a padre_1 →
   indices.
4:     Calcular la distancia de Hamming entre padre_1 y cada elemento de indices
5:     Establecer como padre_2 la solución de indices con la mayor distancia de Hamming a
   padre_1
6: end procedure
```

9.6. Operador Reemplazo: *Crowding* Determinístico

Otra forma de intentar mantener cierta diversidad en la población es mediante la modificación del Operador de Reemplazo. En vez de siempre eliminar las peores soluciones para introducir otras mejores, se sustituirán individuos de la población similares a las nuevas soluciones si estas últimas son mejores. Esto da lugar que se permitirá mantener las peores soluciones en la población siempre que estén aportando diversidad a la población.

Este operador [30] viene representado en el Algoritmo 29. En definitiva, el objetivo de este operador será comprobar si las nuevas soluciones (hijas) son mejores que las soluciones de la población actual más cercanas a ellas. Si las soluciones hijas constituyen una mejora, entonces sustituirán a las soluciones más cercanas en la población.

Algorithm 29 Operador de Reemplazo por Cercanía

```
1: procedure (Reemplazo Cercanía)
2:     for i in 0..numHijos do
3:         Calcular distancia de Hamming de hijo_i a todos los elementos de la población
4:         Elegir la solución con la menor distancia de Hamming a hijo_i → solucion
5:         if valor_solucion <valor_hijo_i then
6:             Sustituir solucion por hijo_i
7:         end if
8:     end for
9: end procedure
```

9.6.1. Reemplazo intensivo

En esta versión, adicionalmente a modificar el operador de reemplazo de forma que las nuevas soluciones sustituirán a las más parecidas siempre que las mejoren, se compararán el resto de soluciones similares (tienen una distancia de Hamming menor que determinado umbral). La idea de esta versión es aumentar aún más la diversidad mediante la eliminación de soluciones peores parecidas a aquellas que ya tenemos.

Si se logra introducir la nueva solución en la población se comprobará si hay más soluciones cercanas. En caso negativo no se hará nada más al respecto. Sin embargo, en caso positivo, nos quedaremos únicamente con la mejor solución entre todas las consideradas (la nueva y las parecidas en la población), eliminando el resto de la población. Como el tamaño de la población debe ser constante, estas vacantes serán cubiertas por soluciones totalmente aleatorias que no sean parecidas a la solución que se ha conseguido mantener en la población. Este funcionamiento viene representado en 30.

Algorithm 30 Operador de Reemplazo por Cercanía intensivo

```
1: procedure (Reemplazo Cercanía)(δ_H)
2:     for i in 0..numHijos do
3:         Almacenar los individuos de la población tal que su distancia de Hamming a hijo_i sea
   menor que δ_H → indCercanos
4:         Calculamos el elemento de indCercanos con mayor valor
5:         El resto de elementos de indCercanos se sustituye por soluciones generadas aleatoria-
   mente
6:         while La solución generada aleatoriamente sea cercana a una existente do
7:             Sustituirla por otra solución generada aleatoriamente
8:         end while
9:     end for
10: end procedure
```

Capítulo 10: Evolución en la propuesta

En este capítulo se realizará un análisis completo paso por paso del proceso de desarrollo del algoritmo. Se mostrarán distintas tablas y gráficas comparativas y sus interpretaciones con el fin de justificar la motivación de las distintas modificaciones introducidas explicadas en el capítulo anterior.

10.1. Diseño Experimental

10.1.1. Criterio de Parada

Se han elegido las instancias mencionadas anteriormente ((QKP) *instances*) ya que también se proporcionaban algunos resultados de otros algoritmos, por lo que resultaba conveniente a la hora de comprobar si los resultados obtenidos por nuestros algoritmos bases eran competitivos. Ya que, en el caso de que no lo fuesen, tendríamos que buscar otros algoritmos base sobre los que trabajar. Sin embargo, nos encontramos con un problema, esto es, el criterio de parada presentado en estos casos es el tiempo (se ha tomado como referencia los experimentos realizados en [41]). Adicionalmente, el tiempo de parada también depende del número de elementos, n, de forma que se tiene:

- Para $n = 100$, se tienen **5 segundos** de ejecución.
- Para $n > 100$, en nuestro caso, $n = 200$ y $n = 300$, se tienen **30 segundos** de ejecución.

Como se ha indicado antes, utilizar el tiempo de ejecución como criterio de parada resulta un problema. Esto se debe a que no es un criterio de parada fiable, ya que depende de la capacidad de computación de cada ordenador. No es comparable la velocidad de los ordenadores actuales con la velocidad de los ordenadores de dentro de 10 años, de la misma no podemos comparar el rendimiento de un ordenador de hace 10 años con respecto a uno actual. E incluso dentro de los ordenadores de la misma generación, dependerá de las características propias de cada computador. Por lo que se llega a la conclusión de que si se quiere que los resultados obtenidos en este trabajo puedan ser usados como referencia, o incluso si se quiere recrear el trabajo, en un futuro se debe cambiar el criterio de parada a algo portable, a algo que sea independiente de cuándo se produzca el experimento. Por ello, se ha decidido cambiar el criterio de parada a un número de iteraciones máximo. Este criterio sí es portable, ya que independientemente de qué tipo de computador se utilice para obtener los resultados siempre se van a obtener los mismos resultados utilizando los mismos parámetros.

Primero debemos indicar lo que entenderemos por iteraciones. Denotaremos como iteraciones al número de veces que se repite el procedimiento completo un determinado algoritmo, en nuestro caso se va a traducir en lo siguiente:

- Para el algoritmo *Random*, el número iteraciones será el número de veces que generemos una solución de forma aleatoria.

- Para los algoritmos genéticos, una iteración puede suponer un número distinto de evaluaciones. En los algoritmos genéticos al número de iteraciones se llama también generaciones.

Dicho esto, para elegir el número de iteraciones máximo que se iba a utilizar se tomó como referencia el tiempo establecido anteriormente. En tanto se tenía elegir un número de iteraciones como criterio de parada, se decidió estudiar a qué equivalía actualmente el criterio de parada por tiempo establecido. Para ello, mediante una variable `contador`, se ejecutaron todos los archivos con el tiempo como criterio de parada y se mostraba por pantalla el número de iteraciones que se había alcanzado. Tras realizar una media de todos estos datos y redondearlo, obtenemos que el nuevo criterio de parada es **90000 iteraciones** para todos los archivos. El que el número de iteraciones máximo no dependa del número de elementos como lo hacía el tiempo de ejecución máximo se puede justificar en tanto que se necesita más tiempo para realizar todos los cálculos si aumenta el número de datos.

Además, para asegurarnos que realmente ambos criterios de parada eran equivalentes, se ejecutaron todos los archivos usando el algoritmo base con criterio de parada por iteraciones y por tiempo. Nótese que no solo se almacenan las soluciones finales, si no que también se almacenan las soluciones intermedias llegado a ciertos porcentajes de la ejecución. Tras esto, se compararon ambos resultados y se pudo comprobar que, efectivamente, eran equivalentes.

Por lo tanto, se puede decir con seguridad que los cambios que apliquemos a un criterio de parada en específico se puede traducir a un cambio en el otro criterio de parada. Esto cabe la pena destacarlo ya que, recordemos, el objetivo de este trabajo es crear un algoritmo útil y competitivo para tratar con problemas *expensive*, por lo que queremos obtener buenos resultados en un tiempo muy reducido.

Para simular de forma eficiente y suficiente esta reducción de tiempo, se propuso el siguiente cambio con respecto al tiempo como criterio de parada:

- En vez de ejecutar los archivos con $n = 100$ durante 5 segundos, lo reduciremos a 25ms.

- En vez de ejecutar los archivos con $n > 100$ durante 30 segundos, lo reduciremos a 150ms.

Realizamos un cálculo básico para comprobar qué proporción de la ejecución estaremos realizando con estos nuevos tiempos:

$$\frac{25 \cdot 100ms}{5 \cdot 1000ms} = 0.5 \qquad \frac{150 \cdot 100ms}{30 \cdot 1000ms} = 0.5$$

Por lo tanto, obtenemos que solo utilizaremos el 0.5 % inicial de cada ejecución, lo que podemos traducir en que nuestro nuevo criterio de parada serán **450 iteraciones**.

Con el fin de comprobar que esta reducción había sido suficiente y necesaria, se ejecutan de nuevo todos los archivos con el algoritmo base ahora con 450 iteraciones y comparamos los resultados con los obtenidos anteriormente con 90000 iteraciones. Podemos ver que, efectivamente, se han obtenido resultados mucho peores.

Finalmente, ya hemos establecido nuestro criterio de parada escalable y tenemos unos resultados base que utilizar. A partir de esto, nuestro objetivo será mejorar estos resultados lo máximo posible.

10.1.2. Parámetros

La ejecución del programa no requiere de la introducción manual de ningún tipo de parámetro. Todos los parámetros que se nombren a continuación vienen definidos dentro del código del programa `main` como constantes globales o que dependen del propio problema.

En primer lugar, recordemos brevemente los parámetros que utilizaremos y sus valores. Estos parámetros no necesitan ser introducidos manualmente al ejecutar el programa, si no que están definidos como constantes en el código:

- `NEVALUACIONESMAX`: Es el número de iteraciones máximas, mantendremos su valor a 450 por lo explicado en el apartado anterior.
- `NTRIES`: Para eliminar la aleatoriedad que proviene de la ejecución con una determinada semilla u otra, cada algoritmo se ejecutará `NTRIES` número de veces, haremos la media de sus ejecuciones y este valor será el resultado que se guardará y se comparará posteriormente con el resto. Para que los resultados puedan ser comparados estadísticamente, pero a la vez se mantenga un tiempo de ejecución razonable, estableceremos su valor a 50.
- `INITSEED`: En relación con la constante anterior, es necesario no utilizar siempre la misma semilla de aleatoriedad, ya que eso resultaría en obtener siempre los mismos resultados, por lo que ejecutarlos varias veces sería una pérdida de tiempo. Así pues, tendremos que usar distintas semillas para cada ejecución. Para asegurar que los resultados sean replicables deben definirse todas estas semillas y que se utilice el mismo conjunto de semillas para todos los algoritmos (para que aún dentro de lo que cabe, todos tengan las mismas condiciones de aleatoriedad). Por conveniencia, en vez de definir todas las semillas a mano, estableceremos manualmente una única semilla inicial que se encargará de generar el resto de semillas que son las que realmente se usarán. En nuestro caso, se ha elegido el valor 5.
- `numcro`: Es el tamaño de la población de individuos que vamos a considerar en los algoritmos. En tanto que el objetivo es obtener buenos resultados en pocas iteraciones, no nos podemos permitir tener un tamaño de población excesivamente grande; ya que a mayor población, más evaluaciones deberán realizarse por generación y esto dará lugar a que se realicen muy pocos cambios de generación (lo que se traduce en pocas iteraciones del algoritmo). Tampoco podemos tener un tamaño de población excesivamente pequeño, ya que resulta un inconveniente el trabajar con muy pocos individuos y este problema no es el objetivo de este trabajo. Por lo tanto, estableceremos su valor a 10.
- `probc`: Es la probabilidad de cruce en los AGs. No lo usamos como parámetro propiamente en el código, ya que para los algoritmos de referencia (AGE y CHC) tienen un número de cruces que se tienen que producir establecidos; en el caso de AGE solo se producirá un cruce por generación y para el CHC se intentarán cruzar todos los individuos posibles.
- `probm`: Es la probabilidad de mutación en los AGs. Recuérdese que para el CHC no se requiere de esta mutación ya que obtiene la diversidad de otros operadores propios. En la literatura se coincide que el valor de este parámetro debe ser uno bastante pequeño, generalmente entre un 1 % y un 10 % de la población. Para asegurarnos que se produzca una mutación por cambio de generación, establecemos su valor a 0.1.
- `EvaluacionMAX`: Es el número de iteraciones/generaciones máximas. Establecemos su valor a 450 por lo ya explicado en el apartado "Diseño Experimental".

- `seed`: Como se ha indicado anteriormente, cada ejecución individual de un algoritmo tendrá asociada una semilla para determinar la aleatoriedad de los acontecimientos y que sea replicable. Este valor se tomará de aquellos que hayan sido generados por `INITSEED`.

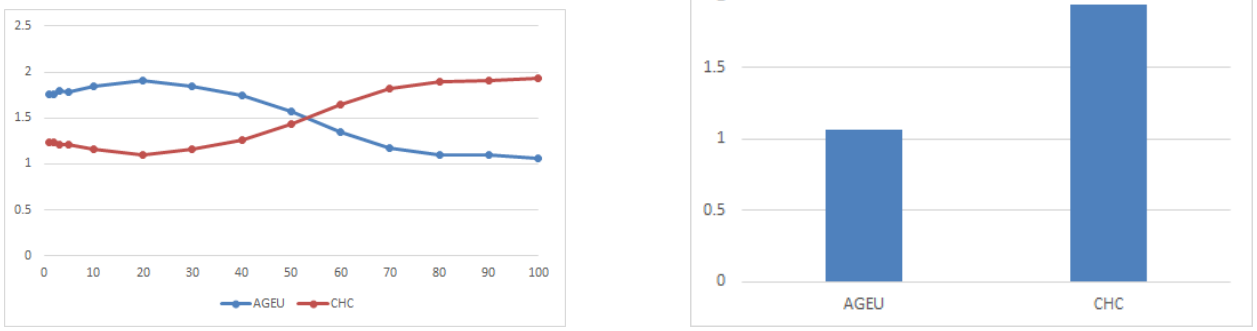
En definitiva, a modo de resumen, podemos rápidamente visualizar en la Tabla 10.1 los distintos parámetros con los que trabajaremos, cuáles son sus usos y sus valores.

Tabla 10.1: Resumen de parámetros utilizados

Parámetros	Resumen	Valor
`NEVALUACIONESMAX`	Criterio de parada: Número de Iteraciones Máximas para cada algoritmo	450
`NTRIES`	Número de veces que se va a ejecutar el algoritmo por cada archivo	50
`INITSEED`	Semilla inicial para poder generar el resto de semillas que utilizaremos para los algoritmos	5
`EvaluacionMAX`	Criterio de parada: Número de Iteraciones Máximas dicho algoritmo	`NEVALUACIONESMAX`
`seed`	Semilla de aleatoriedad para determinada ejecución del algoritmo	Valor generado aleatoriamente por `INITSEED`
`numcro`	Número de elementos de la población de soluciones	10
`pmut`	Probabilidad de mutación de las soluciones de la población ([0,1])	0.1
`probc`	Probabilidad de cruce de los individuos de la población	- AG: 2 individuos por Torneo - CHC: Todos los posibles sin repetición

Es importante mencionar que para todas las ejecuciones se han almacenado no solo los resultados finales, sino las mejores soluciones que se han encontrado en diversos momentos de la ejecución (*milestones*). Es decir, se han almacenado las mejores soluciones encontradas en los porcentajes de ejecución $\{1, 2, 3, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. Esto se podrá ver claramente cuando se muestren gráficas que representen de forma visual el *ranking* de los distintos algoritmos comparados; sin embargo, por falta de espacio en las páginas y comodidad tanto para la autora como para el lector, en las tablas solo se mostrarán los resultados finales de las ejecuciones, aunque se harán menciones al comportamiento general (para determinar si puede estar convergiendo a una solución de forma prematura).

Tambiés es necesario comentar que los datos tienen distintas escalas, por lo que a la hora de estudiar las diferencias entre los resultados de las *milestones* no podemos simplemente hacer una media de todos los valores. Por ello se ha intentado normalizar los intervalos considerando que el resultado final es el máximo del intervalo y 0 el mínimo (ya que en nuestro problema todos los valores de los elementos, individuales y combinados, son no negativos). Por tanto, en las tablas



(a) Ranking durante todas las *milestones*

(b) Ranking final

Figura 10.1: Comparación de AGEU y CHC

de diferencias y sus respectivas gráficas se ven representados los porcentajes de dichos intervalos asociados a las diferencias entre el resultado final y el resultado de cada *milestone*.

Todos los códigos y resultados pueden ser encontrados en el respositorio de Github de este trabajo[1].

10.2. Resultados versión expensive

Tal y como hemos mencionado en capítulos anteriores, el objetivo de este trabajo es el desarrollo de un algoritmo que ante todo sea capaz de obtener resultados competentes en poco tiempo, lo que se puede traducir en un número pequeño de iteraciones. Por lo que el siguiente paso será obtener los resultados para los mismos algoritmos que en la sección anterior, pero para su versión *expensive*, es decir, utilizaremos un número de iteraciones máximas de 450, tal y como hemos establecido en los parámetros.

Obtener resultados para esta versión es fundamental, ya que nos aporta las bases sobre las que trabajaremos y necesitaremos mejorar todo los posible durante este desarrollo. También es interesante observar lo mucho que empeoran los resultados frente a su versión normal, lo cual era de esperar, ya que recordemos que solo estamos utilizando el primer 0.5 % de las iteraciones de su versión original.

Por ello, en la tabla A.1, se muestran los resultados de la ejecución del AGE, mientras que en la tabla A.2, se muestran los resultados de la ejecución del CHC.

Cabe mencionar que originalmente el único algoritmo de referencia que se consideró era el AGE. Sin embargo, durante el desarrollo se consideró la posibilidad de introducir elementos propios del algoritmo CHC, ya que es un algoritmo bastante balanceado que suele dar buenos resultados. Por ello, también se realizó una comparación entre los resultados obtenidos por el AGE y los del CHC. Esta comparación se puede apreciar en la gráfica 10.1.

También podemos apreciar en la tabla 10.2 la media de las distancias entre las soluciones obtenidas en cada *milestone* en la ejecución de AGE con la final (también más visible en la gráfica 10.2).

[1] https://github.com/Itrilor/TFG

Tabla 10.2: Diferencias de los resultados de las distintas milestones con respecto a la final para el AG

AGE 450								
n	milestone	Diferencia	n	milestone	Diferencia	n	milestone	Diferencia
100	1	20.5989	**200**	1	27.5017199	**300**	1	28.8527814
	2	16.3468		2	22.9143937		2	23.90292863
	3	14.0108		3	20.5848898		3	21.25337355
	5	10.5074		5	16.5204328		5	17.2023095
	10	5.68841		10	10.4844166		10	11.26211424
	20	2.33124		20	5.40382962		20	6.370632961
	30	1.1489		30	3.14660193		30	4.193702678
	40	0.6321		40	1.87304152		40	2.913529408
	50	0.36445		50	1.10940335		50	1.986383412
	60	0.20967		60	0.64701591		60	1.330223249
	70	0.12326		70	0.3791851		70	0.852926436
	80	0.07038		80	0.20608454		80	0.475363912
	90	0.03592		90	0.08155519		90	0.198954275

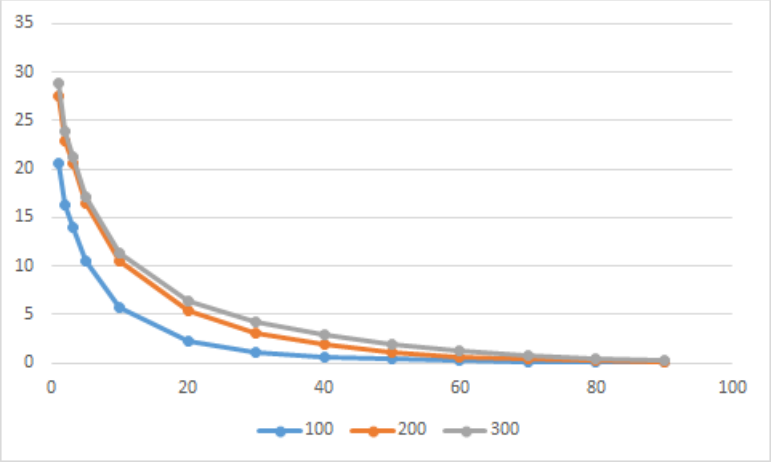


Figura 10.2: Gráfica asociada a la tabla 10.2

Tabla 10.3: Diferencias de los resultados de las distintas milestones con respecto a la final para el algoritmo CHC

CHC 450								
n	milestone	Diferencia	n	milestone	Diferencia	n	milestone	Diferencia
100	1	12.8016	**200**	1	19.8850218	**300**	1	31.0012675
	2	9.87116		2	17.2722142		2	29.0482187
	3	7.03101		3	15.6907262		3	27.939129
	5	4.13105		5	13.1825268		5	25.9419819
	10	0.66658		10	5.55650253		10	18.8008051
	20	0.00171		20	0.36295397		20	2.2348952
	30	2.5E-05		30	0.00250816		30	0.15429962
	40	0		40	0.00030503		40	0.00179232
	50	0		50	6.9674E-05		50	7.5388E-05
	60	0		60	0		60	0
	70	0		70	0		70	0
	80	0		80	0		80	0
	90	0		90	0		90	0

De la misma forma, en la tabla 10.3 se puede apreciar la media de las diferencias entre las soluciones obtenidas en cada *milestone* de la ejecución del algoritmo CHC con la final (también más visible en la gráfica 10.3).

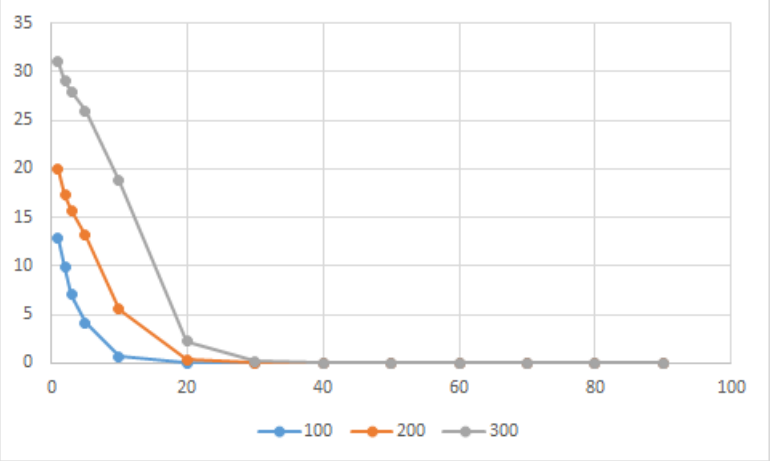
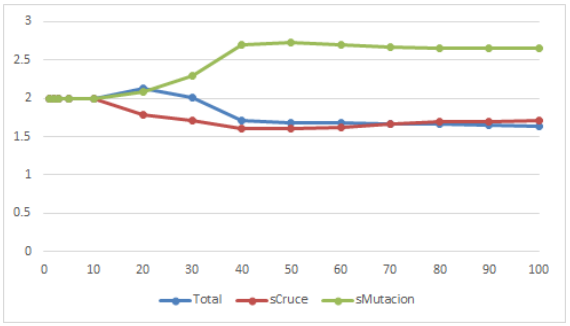


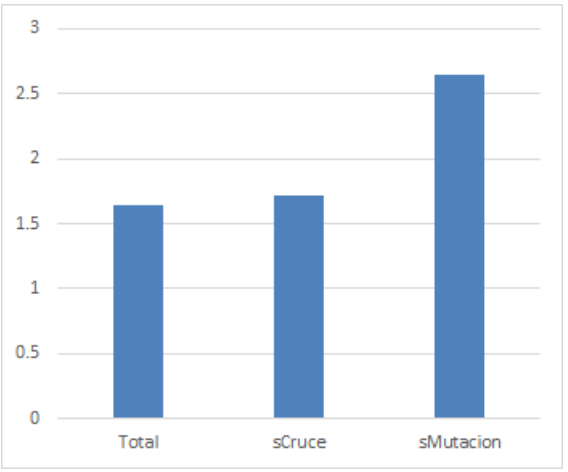
Figura 10.3: Gráfica asociada a la tabla 10.3

Por lo que se puede ver en la tabla 10.2 no se puede decir que converja prematuramente, de hecho en todo momento realiza mejoras significativas. Esto se puede observar en el hecho de que las diferencias a la última solución van disminuyendo en gran medida en cada una de las *milestones*.

Sin embargo, en la tabla 10.3 se tiene que las soluciones convergen rápidamente a la solución final antes de llegar siquiera el 50 % de la ejecución total. De esto se puede inferir que los mecanismos de aumento de la diversidad de CHC no son de mucha utilidad en nuestro caso. También se debe tener en cuenta que estamos usando una modificación de CHC donde no se reinicia la población, ya que se prevee su rápida convergencia, lo que causaría una gran cantidad de reinicios y, por tanto, evaluaciones, las cuales estamos intentando mantener al mínimo. En contrapartida a no usar reinicio de población, se mantiene la mutación de las soluciones con el fin de aportar más diversidad. A



(a) Ranking durante todas las *milestones*



(b) Ranking final

Figura 10.4: Comparación de las distintas versiones del histórico

vista de los resultados se puede entender que esto no es suficiente para mantener la diversidad.

Por otra parte, si nos fijamos en la gráfica 10.1 se puede ver cómo el algoritmo CHC es mejor que el AGE mientras que se siguen produciendo cambios, es decir, antes de que se produzca el 50 % de la ejecución. Por lo que se considerará posteriormente una mezcla de ambos algoritmos, con el fin de aprovechar los buenos resultados iniciales del CHC y la capacidad de no converger rápidamente que presenta AGE con el fin de tener un buen punto de partida.

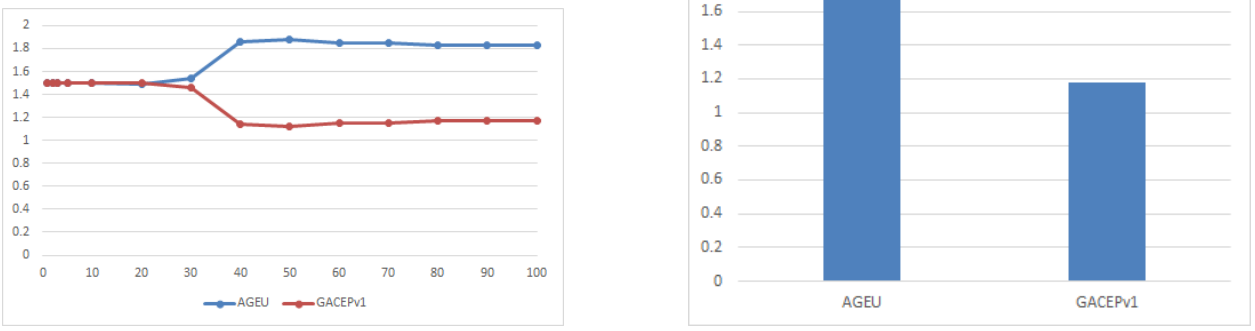
10.3. Incorporación del histórico

La primera modificación que se propone es la introducción de un histórico 22, explicado en la primera sección del capítulo anterior. La principal motivación de esta modificación es incidir en la explotación de las buenas soluciones, ya que se almacenarán los elementos más predominantes en las mejores soluciones y en las peores, con el fin de reincidir con mayor frecuencia en los primeros y rehuir de los segundos.

Obviamente, este histórico deberá tener un proceso de aprendizaje (en el que almacena todas las soluciones distintas para luego discernir cuáles son las mejores y cuáles las peores y sus respectivos elementos) y un proceso de reacción, en la que pondrá en práctica la información aprendida en la fase anterior. Con el fin de que no sea un único procedimiento y pueda ir aprendiendo a lo largo de la ejecución (para intentar evitar quedarse en máximos locales), estas dos fases se irán turnando cada 50 generaciones.

Para poner a prueba su utilidad, se estudian los resultados obtenidos de aplicar el histórico solo al operador de reparación (si tiene que añadir elementos, elegirá los mejores; si tiene que eliminar elementos, elegirá los peores), solo a la mutación (tiene prioridad insertar mejores elementos y eliminar peores) y a ambos a la vez. Los resultados de este estudio se muestran en la tablas A.3, A.4 y A.5 y, más visualmente, en la gráfica 10.4.

En esta gráfica 10.4 podemos apreciar que las mejores opciones son aplicarlo solo al cruce o a ambos; de hecho, por muy poco se tiene que aplicar ambas modificaciones es lo mejor. El uso de ambas opciones también viene justificado por un intento de mantener la consistencia, y porque es posible que en algún momento del desarrollo se puedan realizar más modificaciones a la mutación



(a) Ranking durante todas las *milestones*

(b) Ranking final

Figura 10.5: Comparación de los resultados del AGE y GACEPv1

de las que el histórico se podría aprovechar.

Además, que cuando solo se aplique el histórico a la mutación de los peores resultados con diferencia es una primera indicación de que la introducción del histórico ha supuesto una mejora al modelo original. Esto se debe a que realmente la mutación se produce un número muy bajo de veces y no suele producir grandes cambios, por lo que si es lo único que se modifica, es muy probable que no se distancie mucho de los resultados originales. Para confirmar esta idea y poder asegurarnos de que se ha dado un paso en una dirección correcta, comparamos en la gráfica 10.7 los resultados originales y los de aplicar el histórico a ambos operadores.

Con esta gráfica 10.7 quedan confirmadas nuestras suposiciones. Es importante mencionar que en sus primeras iteraciones ambos algoritmos obtienen los mismos resultados ya que se comportan de la misma forma. Recordemos que cada fase del histórico ocupa 50 iteraciones, por lo que durante las primeras 50 iteraciones va a mantener el mismo comportamiento que AGE, con la excepción de que va a ir almacenando las soluciones para su posterior análisis. Estas 50 iteraciones suponen un poco más del 10 % inicial de las iteraciones totales, por lo que es obvio que para todas estas *milestones* sus resultados van a ser iguales. También cabe la pena mencionar que en el momento en el que difieren los resultados de los algoritmos, su clasificación se mantiene casi constante, por lo que queda claro que el uso del histórico hace que el algoritmo sea consistentemente mejor.

Es más, con esta modificación podemos crear las bases de lo que será nuestro nuevo algoritmo, al cual llamaremos **GACEP** (***Genetic Algorithm for Combinatory Expensive Problem***). Para establecer una notación común entre las distintas modificaciones, todas se nombrarán de la forma "GACEPvx", donde `x` se sustituirá por el número de la versión. En este caso, el introducir el histórico al AG lo llamaremos **GACEPv1**.

Los resultados de la ejecución de este algoritmo vienen recopilados en la tabla A.5.

A continuación haremos lo mismo que para los algoritmos de referencia y observaremos si se ha producido algún tipo de convergencia prematura. Esto se puede ver en la tabla 10.4 y en su gráfica asociada 10.6.

En la tabla 10.4 se puede ver que GACEP tiene un comportamiento similar con respecto a la progresión de las mejores soluciones a lo largo de la ejecución que el algoritmo base AGE. Se observa que no se produce ninguna convergencia prematura, aunque es cierto que en la segunda mitad de las evaluaciones se producen muchos menos cambios que en el resto.

Tabla 10.4: Diferencias de los resultados de las distintas milestones con respecto a la final para el algoritmo GACEPv1

GACEPv1								
n	milestone	Diferencia	n	milestone	Diferencia	n	milestone	Diferencia
100	1	24.8702	200	1	28.9265669	300	1	28.4705381
	2	20.8839		2	24.473429		2	23.6064829
	3	18.6934		3	22.2164778		3	21.0057888
	5	15.4131		5	18.2724015		5	17.0150669
	10	10.9019		10	12.4140872		10	11.1442233
	20	5.64471		20	8.14923733		20	6.84098075
	30	4.10969		30	5.35892582		30	4.4650068
	40	1.43438		40	2.30317627		40	2.05939817
	50	0.63711		50	0.97838195		50	1.03637898
	60	0.34425		60	0.61345105		60	0.65967857
	70	0.18073		70	0.22987072		70	0.32037747
	80	0.09555		80	0.17789507		80	0.21069325
	90	-0.0002		90	0.02225991		90	0.04461874

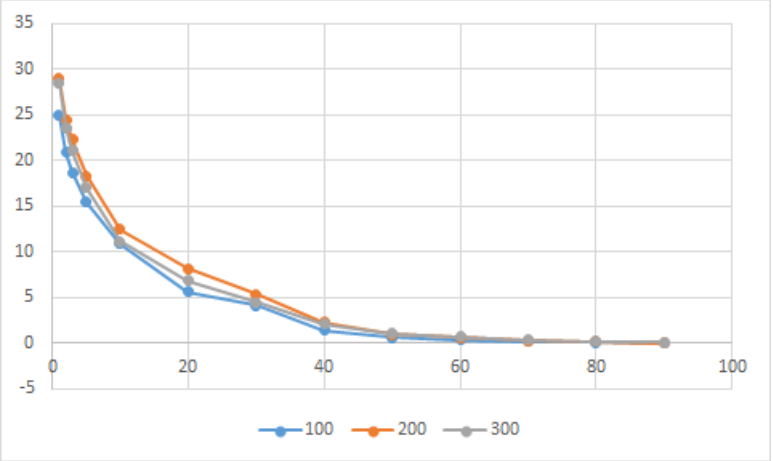


Figura 10.6: Gráfica asociada a la tabla 10.4

También aplicaremos el test estadístico de p-valores de Wilcoxon para determinar si hay diferencias significativas entre AGE y GACEPv1. Este test viene representado en la tabla 10.5.

En dicha tabla 10.5 podemos observar que GACEPv1 es mejor que AGE con una significación menor del 5 %. Esta comparación tiene un error acumulado del 5 %. Por lo que podemos decir con bastante seguridad es que GACEPv1 es, efectivamente, superior a AGE.

10.4. Uso de GRASP

La modificación anterior tenía como objetivo aumentar la explotación de los buenos resultados. Para volver a conseguir un buen balance de explotación y exploración en el algoritmo lo siguiente que debemos pensar es qué clase de modificación se podría introducir de forma que aumentase la diversidad de la población y/o de las soluciones a considerar.

Tabla 10.5: Test de Wilcoxon de AGE Y GACEPv1

	AGE	**GACEPv1**	**Accumulated Error (%)**
AGE	1	4.90E-04	5
GACEPv1	4.90E-04	1	5

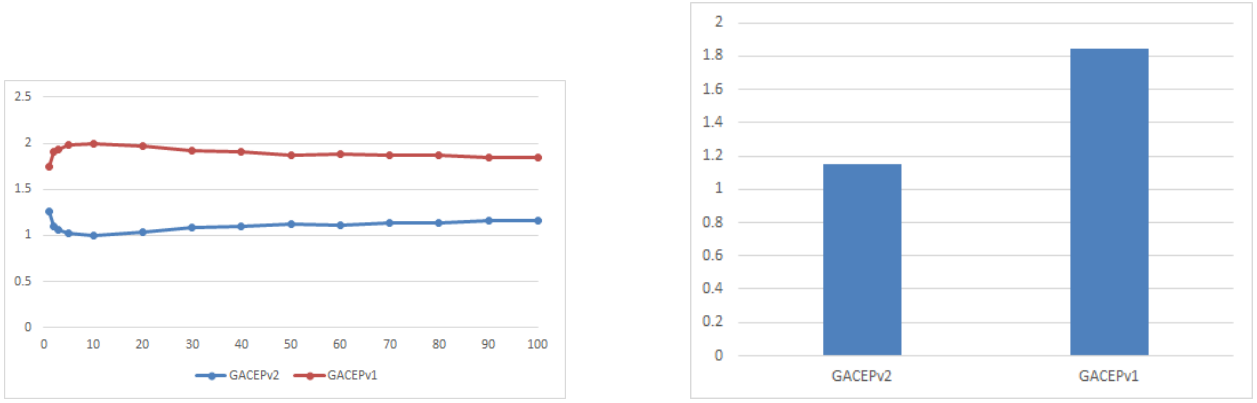
La siguiente modificación que se plantea es la sustitución del uso del algoritmo Greedy por una variante suya: GRASP. Esta modificación solo tiene sentido usarla en el operador de reparación en la fase de aprendizaje del histórico. Esto es porque el único sitio donde se utiliza Greedy es en el operador de reparación, que intenta introducir siempre el mejor elemento a la solución y eliminar el peor. Durante la fase de acción del histórico ya realiza de por si un procedimiento similar al del GRASP 25, ya que le da prioridad a introducir alguno (sin comprobar si es verdaderamente el mejor) de los mejores elementos aprendidos y eliminar alguno (sin comprobar si verdaderamente es el peor) de los peores elementos aprendidos.

El objetivo de esta modificación es introducir un operador que permita diversificar la población, eliminando el determinismo de siempre elegir las mismos elementos. De esta forma, somos capaces de darle más importancia a la parte de exploración del algoritmo, pero siempre manteniendo que las soluciones sigan intentando ser lo mejor posible.

Por tanto, **GACEPv2** será resultado de añadir un operador GRASP al operador de reparación de GACEPv1.

Los resultados de la ejecución del algoritmo GACEP con esta modificación pueden verse en la tabla A.6.

Con esto, debemos comprobar si realmente se ha producido una mejora con respecto a la modificación anterior. Para ello compararemos sus resultados y estableceremos un *ranking*, siendo la gráfica correspondiente a esta comparación la gráfica 10.7.



(a) Ranking durante todas las *milestones*

(b) Ranking final

Figura 10.7: Comparación de los resultados del GACEPv1 y GACEPv2

En esta gráfica 10.7 se puede ver cómo la nueva versión, introduciendo GRASP, es consistentemente mejor que la versión anterior. Es más, se puede afirmar que es mejor en casi todos los datos utilizados. Por lo tanto, se procederá a continuar con el desarrollo de nuestro algoritmo sobre esta nueva versión.

A continuación, se mostrará una tabla 10.6 que almacena la media de las diferencias de los resultados de cada *milestone* con respecto a la final, con el fin de comprobar si podemos sacar

información sobre la convergencia de las soluciones en el tiempo.

Tabla 10.6: Diferencias de los resultados de las distintas milestones con respecto a la final para el algoritmo GACEPv2

GACEPv2								
n	milestone	Diferencia	n	milestone	Diferencia	n	milestone	Diferencia
100	1	25.3737	200	1	29.8461881	300	1	28.9145096
	2	20.6761		2	24.9678186		2	24.0603906
	3	18.4198		3	22.561253		3	21.4992992
	5	14.7471		5	18.4329667		5	17.1360467
	10	9.94967		10	11.9316113		10	10.5285922
	20	5.62915		20	7.98414965		20	6.88853082
	30	4.38901		30	5.14003393		30	4.43403031
	40	0.7794		40	2.28877565		40	2.14374372
	50	0.21656		50	1.07903317		50	1.15285329
	60	0.06538		60	0.42600859		60	0.59987399
	70	0.01904		70	0.04685168		70	0.17475741
	80	0.00528		80	0.01998544		80	0.10914333
	90	-0.0014		90	0.00014457		90	0.00313706

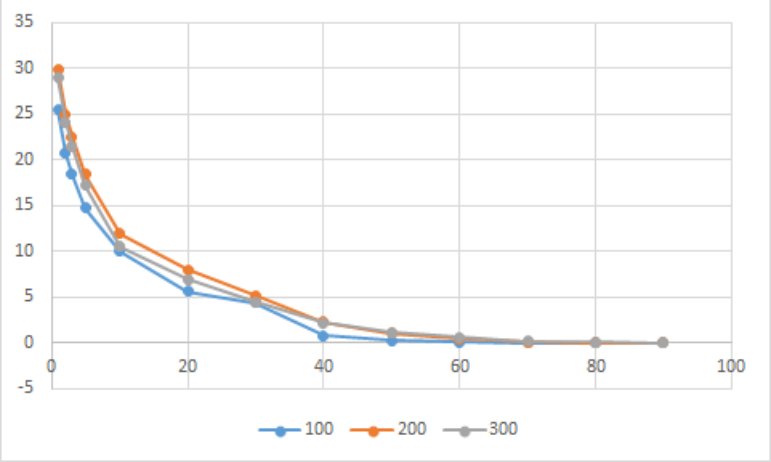


Figura 10.8: Gráfica asociada a la tabla 10.6

Comparando las tablas de diferencias 10.4 y 10.6 se puede ver que al principio parece que sí introduce algo más de diversidad (ya que las soluciones en dichas *milestones* son más diferentes), aunque esto es solo notable para los tamaños $n = 200$ y $n = 300$.

También aplicaremos el test estadístico de p-valores de Wilcoxon para determinar si hay diferencias significativas entre GACEPv1 y GACEPv2. Este test viene representado en la tabla 10.7.

En dicha tabla 10.5 podemos observar que GACEPv2 es mejor que GACEPv1 con una significación menor del 5 %. Esta comparación tiene un error acumulado del 5 %. Por lo que podemos decir con bastante seguridad es que GACEPv2 es, efectivamente, superior a GACEPv1.

Tabla 10.7: Test de Wilcoxon de GACEPv1 Y GACEPv2

	GACEPv2	**GACEPv1**	**Accumulated Error (%)**
GACEPv2	1	1.38E-13	5
GACEPv1	1.38E-13	1	5

10.5. Operador de Cruce Intensivo

La siguiente idea a seguir tiene que ver de nuevo con la explotación de las características de las mejores soluciones. Anteriormente se propuso introducir una modificación para mejorar la explotación en el propio operador de reparación y en la mutación; en este caso, se pretende introducir este operador en el propio operador de cruce directamente.

Esta modificación se basa en que las soluciones hijas heredarán una mayor cantidad de genes del mejor de los dos padres 26. La motivación de este cambio es la generación de nuevas soluciones muy parecidas a las buenas que ya forman parte de la población, con lo que hay una mayor probabilidad de que se sigan desarrollando las soluciones en ese ámbito.

Para este trabajo probaremos con varios porcentajes de elementos que deben ser pertenecientes al mejor padre: 50 %, 60 %, 70 % y 80 %, para decidir cuál sería la mejor proporción. Téngase en cuenta que elegir el 50 % de los genes del mejor padre no tiene por qué tener el mismo resultado que el cruce uniforme que estamos actualmente realizando; ya que la primera opción permite que ambos hijos tengan genes en común (que provengan del mismo padre), mientras que para la segunda opción eso es imposible.

Adicionalmente, consideraremos también dos posibilidades mayores: el utilizar GRASP de nuevo para este tipo de cruce o no. De antemano podríamos suponer que el uso del GRASP dará mejores resultados, ya que ayudará a diversificar unas soluciones hijas que serán muy parecidas entre sí en tanto que tomarán bastantes elementos del mismo padre. Pero esto debemos comprobarlo empíricamente, por ello, se calculan todos los resultados posibles y se comparan entre sí para determinar cuál es la mejor opción, la cual compararemos posteriormente con nuestro GACEPv2.

10.5.1. Utilizando GRASP

La lógica de utilizar GRASP para el cruce intensivo es la misma que para el cruce uniforme, la utilizaremos en el operador de reparación tras generar los hijos resultantes de dicho cruce.

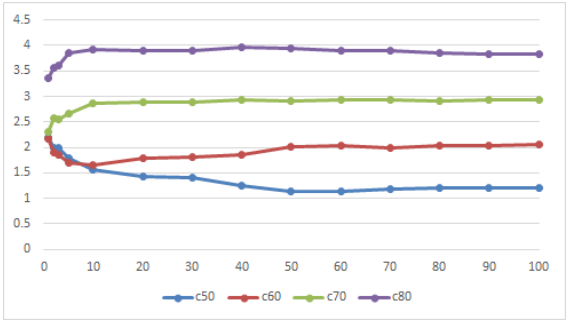
Estas versiones recibirán el nombre "**GACEPv3cx**", donde `x` en este caso representaría el porcentaje de elementos que toma del mejor padre. Los resultados de cada ejecución vienen almacenados en las tablas A.7, A.8, A.9 y A.10.

Dicho esto, podemos ver en la gráfica 10.9 el *ranking* de los distintos porcentajes asociados al operador de cruce.

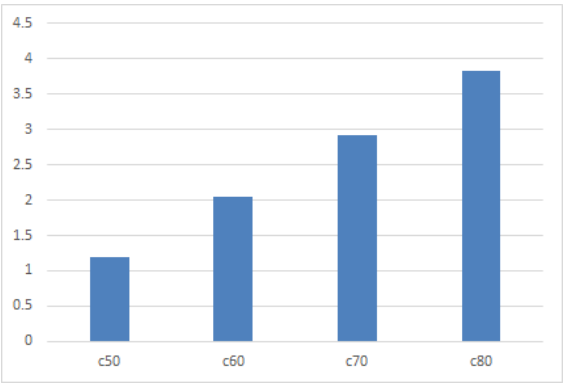
10.5.2. Sin utilizar GRASP

En este caso se seguiría usando el Operador de Reparación original (15), donde se elige automáticamente el mejor elemento posible para introducir y el peor elemento posible para eliminar.

Estas versiones recibirán el nombre "**GACEPv3cxwo**", donde `x` en este caso representaría el porcentaje de elementos que toma del mejor padre y "wo" indica que no se está utilizando GRASP



(a) Ranking durante todas las *milestones*

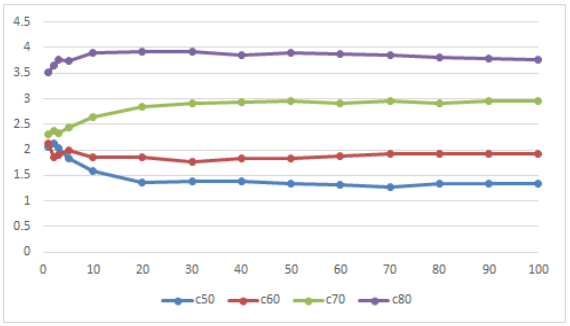


(b) Ranking final

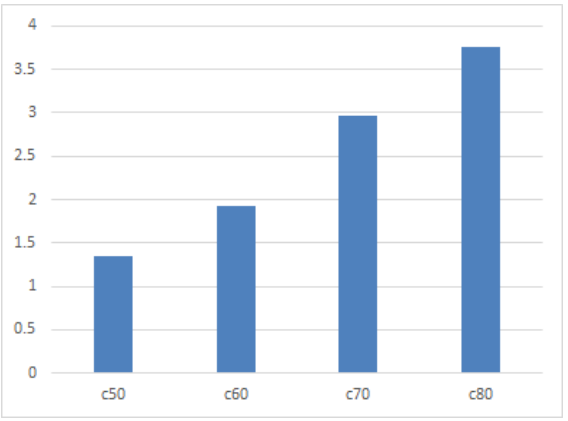
Figura 10.9: Comparación de los resultados de los GACEPv3cx

(*without GRASP*). Los resultados de cada ejecución vienen almacenados en las tablas A.11, A.12, A.13 y A.14.

En la gráfica 10.10 podemos apreciar el *ranking* de los distintos porcentajes asociados al operador de cruce.



(a) Ranking durante todas las *milestones*



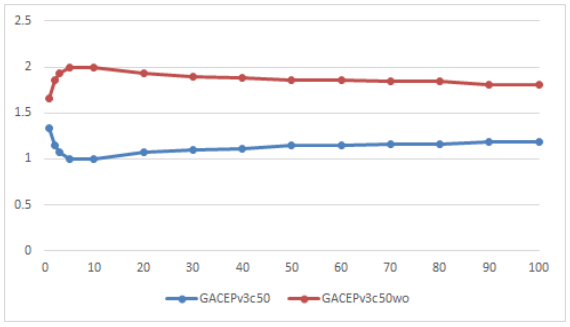
(b) Ranking final

Figura 10.10: Comparación de los resultados de los GACEPv3cxwo

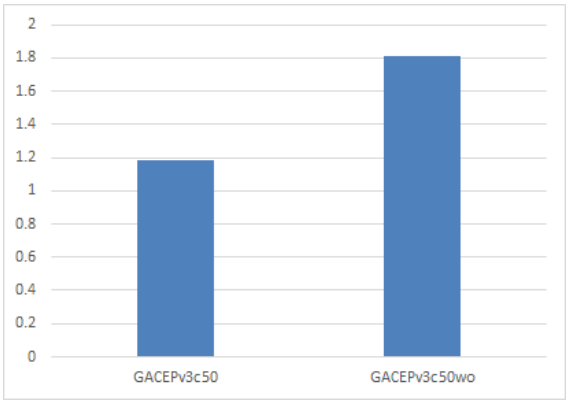
Se puede apreciar que en ambos casos, el comportamiento que tienen los algoritmos es similar, cuanto mayor sea la intensidad con la que se aplique el nuevo operador de cruce peores son los resultados. Con esto se puede empezar a intuir que, al menos para poblaciones pequeñas, tener muchas soluciones parecidas resulta contraproducente, ya que reduce la diversidad y llega a estancar a la población. Por lo tanto, procederemos a comparar las mejores versiones de ambas opciones: las dos utilizando un 50 % de los elementos del mejor padre. En la gráfica 10.11 podemos apreciar el *ranking* de ambas opciones tanto para las distintas *milestones* como la final sola.

En 10.11a podemos ver que el usar la opción con GRASP es superior a la versión que no lo usa. Con esto podemos extraer que el uso del GRASP es adecuado para ambos operadores de cruce: el uniforme 16 y el porcentual 26.

Esto último nos lleva a tener que comparar los algoritmos GACEPv2 y GACEPv3c50 para determinar si se ha producido alguna mejora. Esta comparación puede ser visualizada en la gráfica 10.12.

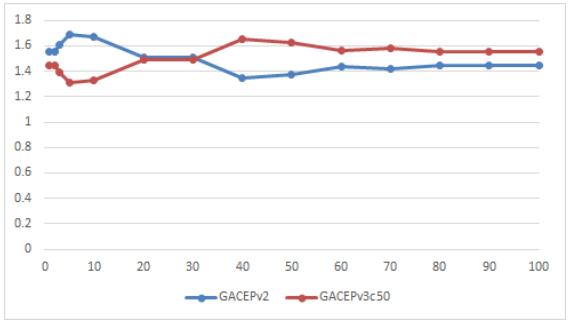


(a) Ranking durante todas las *milestones*

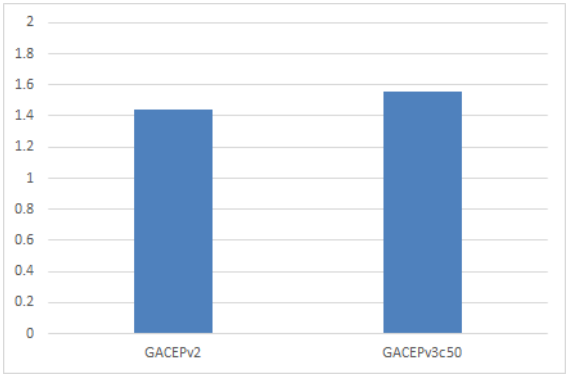


(b) Ranking final

Figura 10.11: Comparación de los resultados de los GACEPv3c50 y GACEPv3c50wo



(a) Ranking durante todas las *milestones*



(b) Ranking final

Figura 10.12: Comparación de los resultados de los GACEPv2 y GACEPv3c50

En la gráfica 10.12a que aunque en las primeras iteraciones el GACEPv3c50 tiene mejores resultados que GACEPv2, esta diferencia se reduce rápidamente. De hecho, las dos versiones tienen resultados muy parecidos, lo cual tiene sentido ya que el cruce uniforme y el cruce porcentual del 50 % son muy similares, aunque no iguales. De cualquier forma, a partir del 30 % de las iteraciones GACEPv2 obtiene mejores resultados que GACEPv3c50, aunque sea por poco. Por lo tanto, ignoraremos esta tercera versión y seguiremos trabajando con la segunda versión como base.

También aplicaremos el test estadístico de p-valores de Wilcoxon para determinar si hay diferencias significativas entre GACEPv2 y GACEPv3c50. Este test viene representado en la tabla 10.8.

Tabla 10.8: Test de Wilcoxon de GACEPv2 Y GACEPv3c50

	GACEPv2	**GACEPv3c50**	**Accumulated Error (%)**
GACEPv2	1	0.494187	5
GACEPv3c50	0.494187	1	5

En dicha tabla 10.5 podemos observar que no se han detectado diferencias significantes entre GACEPv2 y GACEPv3c50 con un error estadístico del 5 %.

10.6. Modificaciones sobre CHC

Llegados a este punto podemos considerar haber hecho algunos avances significativos sobre el AGE y es momento de preguntarse si estas modificaciones se podrían aplicar al algoritmo CHC. Recordemos que en 10.1a podíamos apreciar que CHC obtenía mejores resultados en la primera mitad de las iteraciones y al final resultaba peor dado a que convergía prematuramente. Por lo tanto, en esta sección vamos a comprobar si aplicando los mismos cambios que al AGE se logra mejorar los resultados del GACEP.

En tanto que no es una versión puramente de GACEP, cambiaremos la notación que se le da a "**GACEPCHCvx**", donde `x` representa el número de la versión.

10.6.1. Incorporación del histórico

El uso del histórico 22 para CHC sigue la misma lógica que para el AGE: el aumento de la explotación de los mejores resultados. Si bien a simple vista se podría pensar que el uso del histórico no debería mejorar los resultados ya que nuestro problema principal es que la población converge rápidamente a una solución se debe recordar cómo se clasificaban los distintos elementos:

- `Mejores`: Aquellos elementos que suelen estar presentes en las mejores soluciones, pero no en las peores.
- `Peores`: Aquellos elementos que suelen estar presentes en las peores soluciones, pero no en las mejores.
- `Sin información`: Aquellos elementos que frecuentemente no aparecen en las mejores ni en las peores soluciones o que aparecen en ambas (por lo que no nos aporta ninguna información ya que no se puede considerar un elemento determinista en lo buena que es una solución).

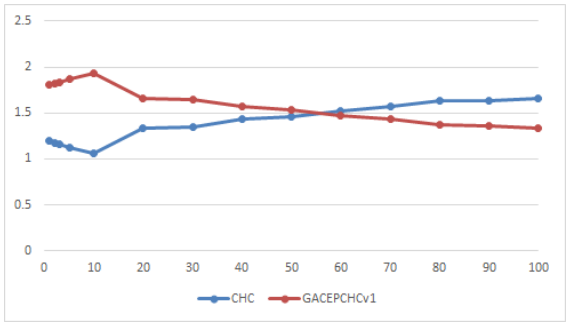
Por lo que si todas las soluciones son muy parecidas entre sí, teniendo todas el mismo conjunto de elementos, estos elementos no se considerarán "mejores" que el resto y tendrán la misma probabilidad de ser elegidas que el resto de elementos que no se han explorado anteriormente.

Esta versión se denominará **GACEPCHCv1**.

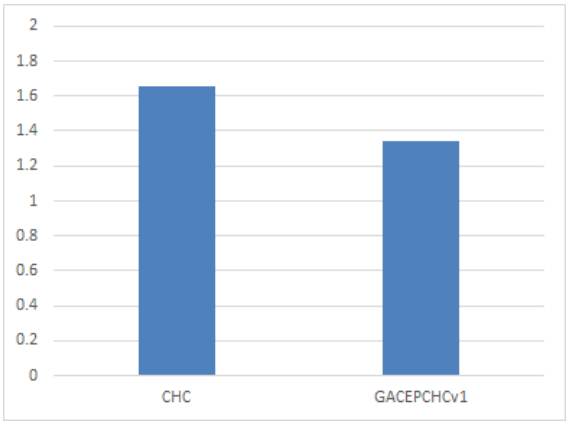
Como hemos hecho previamente, obtendremos sus resultados, los cuales están almacenados en A.15. Estos resultados los compararemos con los del CHC versión *expensive* para determinar si se ha producido algún tipo de mejora. Esta comparación será visible en la gráfica 10.13

En esta gráfica 10.13 se puede apreciar que aunque inicialmente GACEPCHCv1 es bastante peor que CHC, tras la mitad de las ejecuciones logra mejorarlo. Esto podría estar causado por el hecho de que CHC converge rápidamente antes de la mitad de las ejecuciones y que GACEPCHCv1 puede que siga mejorando poco a poco como lo harían las versiones anteriores basadas en AGE debido a cierto aumento de la diversidad en la población. Para comprobar si este es el caso, estudiaremos las diferencias entre las soluciones de las distintas *milestones* y la final. Esta información se mostrará en la tabla y su correspondiente gráfica

Efectivamente, en la gráfica 10.14 se puede comprobar visualmente que tiene un comportamiento parecido al de las versiones de AGE, aunque bien es verdad que es bastante más abrupto.



(a) Ranking durante todas las *milestones*



(b) Ranking final

Figura 10.13: Comparación de los resultados de CHC y GACEPCHCv1

Tabla 10.9: Diferencias de los resultados de las distintas milestones con respecto a la final para el algoritmo GACEPCHCv1

GACEPCHCv1								
n	milestone	Diferencia	n	milestone	Diferencia	n	milestone	Diferencia
100	1	26.7201	**200**	1	25.7172359	**300**	1	26.2254573
	2	23.2096		2	22.3721298		2	22.0835516
	3	21.5554		3	20.5934122		3	20.2131967
	5	18.774		5	17.9674193		5	17.375892
	10	14.7039		10	13.6488991		10	12.7586834
	20	5.48547		20	4.77580753		20	4.55863648
	30	4.59547		30	4.01781911		30	3.702351
	40	2.47563		40	2.54730194		40	2.39425407
	50	1.75038		50	1.97120082		50	1.90218522
	60	1.14161		60	1.35810173		60	1.29008322
	70	0.62625		70	0.82009965		70	0.81355995
	80	0.44086		80	0.55229199		80	0.54620318
	90	0.00726		90	0.01307933		90	0.03623624

10.6.2. Uso del GRASP

Nuestro objetivo sigue siendo aumentar la exploración del algoritmo para conseguir una población aún más diversa y que las soluciones no converjan de forma prematura. Por ello se considera que este algoritmo se podría beneficiar en gran medida del uso del GRASP, podremos mantener la diversidad de la población a lo largo de la ejecución del algoritmo y, a la vez, se garantiza que las soluciones seguirán siendo competentes.

Esta nueva versión está construida usando GACEPCHCv1 de base y la denominaremos **GACEPCHCv2**.

Tras su implementación, debemos recoger sus resultados, los cuales están almacenados en la tabla A.16. Tras esto, en primer lugar compararemos GACEPCHCv1 y GACEPCHCv2 para comprobar que realmente se ha producido una mejora de los resultados con respecto a la versión anterior. Esta comparación será visible en la gráfica 10.15.

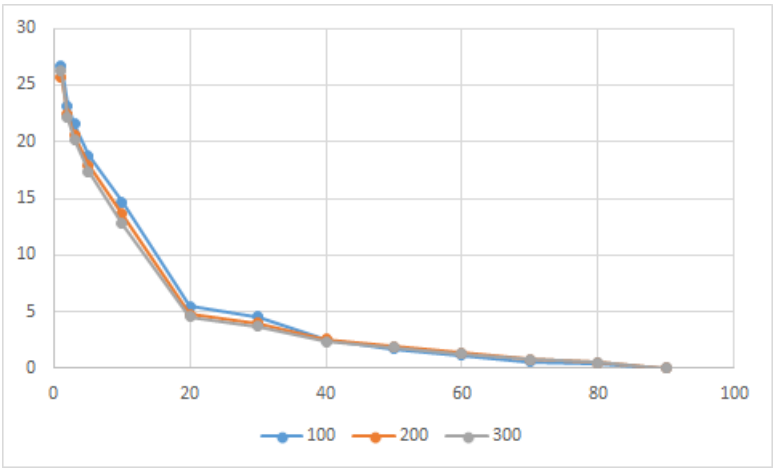
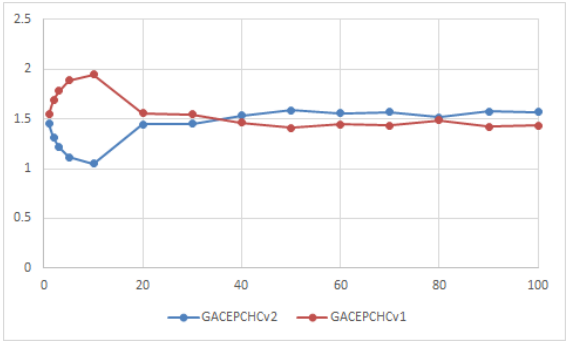
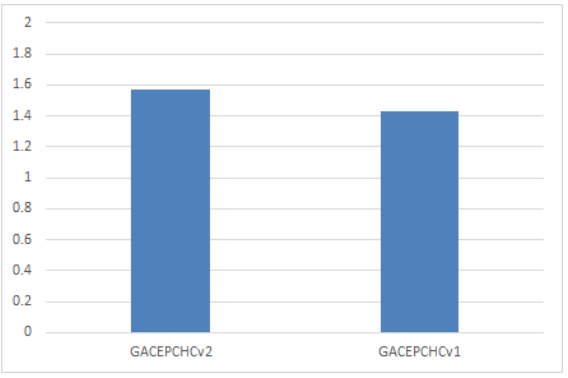


Figura 10.14: Gráfica asociada a la tabla 10.9



(a) Ranking durante todas las *milestones*



(b) Ranking final

Figura 10.15: Comparación de los resultados de GACEPCHCv1 y GACEPCHCv2

En esta gráfica 10.15 podemos comprobar que efectivamente no se produce una mejora con respecto a la versión anterior, aunque los resultados finales son muy cercanos. Por lo que podemos proceder a realizar una comparación de GACEPCHCv1 con la mejor versión hasta el momento para el algoritmo AGE: GACEPv2. Esta comparación será visible en la gráfica 10.16.

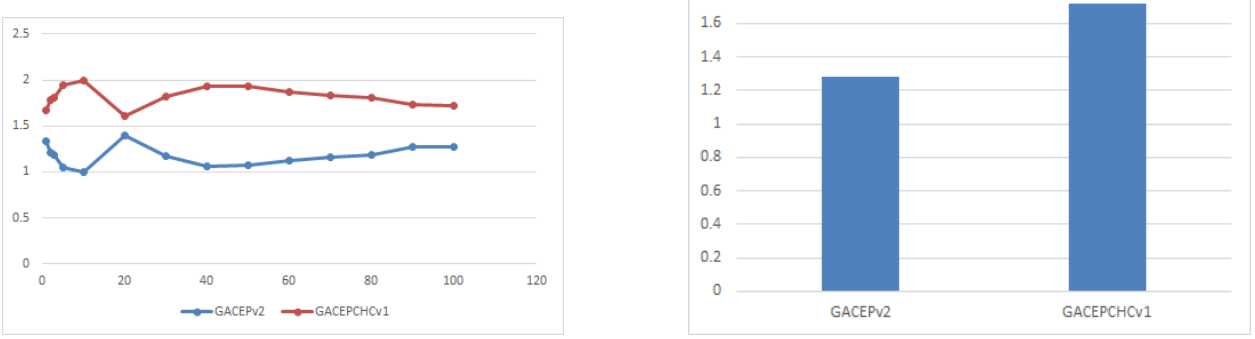
En la gráfica 10.16 se puede ver cómo GACEPv2 obtiene mejores resultados que GACEPCHCv1 en la mayoría de las *milestones*, sobre todo y más importante, obtiene mejores resultados finales. En tanto que no se ha logrado mejorar los resultados de una versión anterior, descartaremos estos cambios y seguiremos trabajando exclusivamente sobre el algoritmo GACEPv2.

También aplicaremos el test estadístico de p-valores de Wilcoxon para determinar si hay diferencias significativas entre GACEPv2 y GACEPCHCv1. Este test viene representado en la tabla 10.10.

Tabla 10.10: Test de Wilcoxon de GACEPv2 Y GACEPCHCv1

	GACEPv2	**GACEPCHCv1**	**Accumulated Error (%)**
GACEPv2	1	6.864E-11	5
GACEPCHCv1	6.864E-11	1	5

En dicha tabla 10.10 podemos observar que GACEPv2 es mejor que GACEPCHCv1 con una



(a) Ranking durante todas las *milestones*

(b) Ranking final

Figura 10.16: Comparación de los resultados de GACEPv2 y GACEPCHCv1

significación menor del 5 %. Esta comparación tiene un error acumulado del 5 %. Por lo que podemos decir con bastante seguridad es que GACEPv2 es, efectivamente, superior a GACEPCHCv1.

10.7. Estudio de la diversidad

Durante este capítulo hemos asociado la abrupta convergencia a una solución como falta de diversidad en la población. Para comprobar esto, se ha decidido utilizar la fórmula para el cálculo de la diversidad propuesta en [42], donde se tiene que:

$$\text{Diversidad}(P) = \frac{1}{n \cdot \texttt{numcro}(\texttt{numcro} - 1)} \sum_{i=1}^{\texttt{numcro}} \sum_{j=1}^{\texttt{numcro}} HD(p_i, p_j) \tag{10.1}$$

donde P es la población, n es la longitud del individuo, HD es la función de la distancia de Hamming y p_i es el individuo i-ésimo de la población P.

Los resultados finales de aplicar la fórmula 10.1 para cada una de las instancias del problema se encuentran en la Tabla A.17. Por otra parte, se han almacenado en la Tabla 10.11 los resultados de aplicar la fórmula 10.1 a la población que hemos conseguido en cada *milestone*.

Como se puede comprobar de forma más visual en la gráfica 10.17, la diversidad de la población se reduce en gran medida al llegar al 20 % de las iteraciones (se reduce a aproximadamente un 10 % de la diversidad inicial). Además, la diversidad en muy pocos casos aumenta, y cuando lo hace, solo aumenta una cantidad inferior a las milésimas. Con esto ya es claro que se pierde mucha diversidad en la población y esto se produce en las etapas iniciales de la ejecución.

Este descubrimiento es lo que nos motiva a implementar nuevas versiones que contengan modificaciones dedicadas especialmente a introducir y/o mantener la diversidad en la población.

10.8. Incrementando la diversidad

Con motivo del estudio realizado en la sección anterior, resulta necesario introducir modificaciones que sean capaces de introducir y/o mantener diversidad en la población. En este trabajo se proponen algunas modificaciones, las cuales se explicarán individualmente y qué sentido podría

Tabla 10.11: Diversidad de la población en las distintas milestones para el algoritmo GACEPv2

GACEPv2								
n	milestone	Diversidad	n	milestone	Diversidad	n	milestone	Diversidad
100	1	0.295980449	**200**	1	0.303604262	**300**	1	0.285172532
	2	0.239986795		2	0.236957486		2	0.228452411
	3	0.20074531		3	0.196295669		3	0.1877086
	5	0.136337387		5	0.127482734		5	0.1235785
	10	0.068669744		10	0.056736476		10	0.051865263
	20	0.034505642		20	0.027917386		20	0.020362715
	30	0.028956809		30	0.025366897		30	0.018587216
	40	0.017469288		40	0.019192113		40	0.01590612
	50	0.011608889		50	0.014284978		50	0.013201013
	60	0.0096147		60	0.012227811		60	0.011819336
	70	0.007890714		70	0.009880155		70	0.010087172
	80	0.007583591		80	0.009224822		80	0.009189166
	90	0.006994988		90	0.008364826		90	0.007948148
	100	0.007128205		100	0.00819632		100	0.007950337

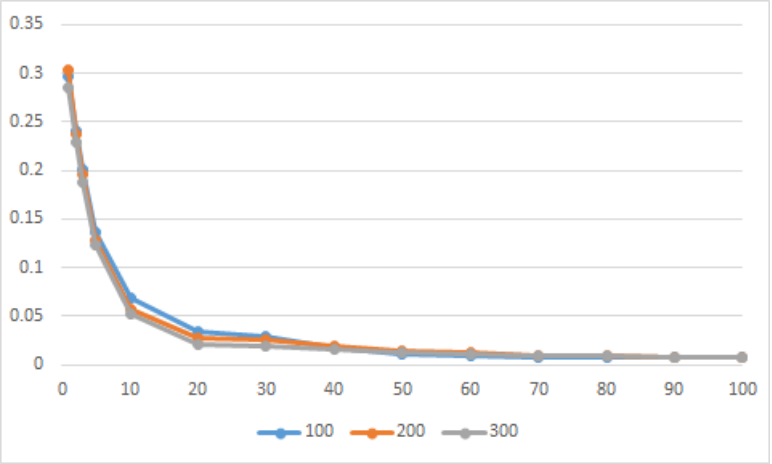


Figura 10.17: Gráfica asociada a la tabla 10.11

tener combinarlas, comparándolas con el algoritmo GACEPv2. Finalmente, compararemos todas las versiones de esta sección entre sí para determinar cuál es la mejor y presentaremos como la versión final de este desarrollo.

Un resumen de las modificaciones que aplicaremos a continuación es:

- Población inicial diversa 27: Nos aseguramos que la población inicial cumpla una condición de diversidad.
- Una versión del operador NAM 28: Alteramos la elección de los padres que se utilizarán para el cruce.
- Una versión del *Crowding* Determinístico 29: Modificamos el operador de reemplazo.

10.8.1. Población Inicial Diversa

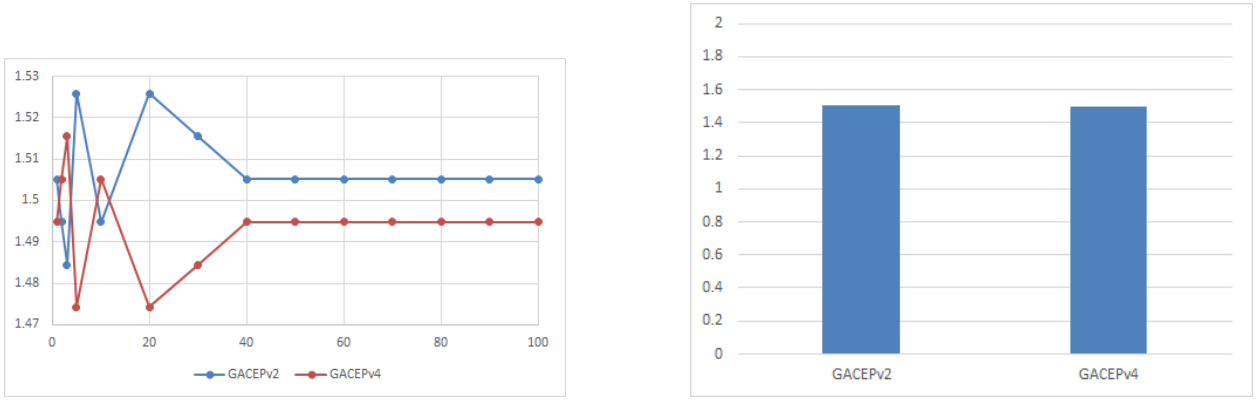
La motivación de esta modificación es asegurar que vamos tenemos con una población lo suficientemente diversa antes de empezar la ejecución del propio algoritmo. La generación de esta población inicial no computa para el número máximo de iteraciones, ya que no nos interesa saber si estas soluciones son buenas o no, solo nos interesa saber que son lo suficientemente distintas.

Consideramos que una población será lo suficientemente diversa cuando cada una de las soluciones que la conforman difiere del resto de soluciones en al menos un `x`% de las variables con las que tratamos. Se iba a realizar un estudio individual para averiguar cuál sería un `x` adecuado, sin embargo, el máximo valor que pudimos ejecutar fue `x`= 5, ya que para valores superiores había archivos en los que era incapaz de encontrar un conjunto de soluciones que satisficiese dichas condiciones.

Por lo tanto, para este trabajo, consideramos que una población es lo suficientemente diversa cuando cada una de las soluciones difiere del resto en, al menos, un 5 % de las variables que la componen.

Esta versión tomará el nombre de **GACEPv4**. Los resultados de este algoritmo se encuentran en la Tabla A.18.

El algoritmo con esta modificación podría no considerarse una versión en sí misma, sino más bien como un prerrequisito para el resto de modificaciones de esta sección. Esto se debe a que la sola modificación de la población inicial no es algo que el algoritmo pueda aprovechar de forma recursiva, es solo una pequeña modificación que se hace al principio para establecer una determinada población. Por lo que su resultado no puede diferir mucho con respecto al de GACEPv2, como se puede apreciar en la gráfica 10.19 (nótese la escala en la que se encuentra el eje Y en 10.18a).



(a) Ranking durante todas las *milestones*

(b) Ranking final

Figura 10.18: Comparación de los resultados de GACEPv2 y GACEPv4

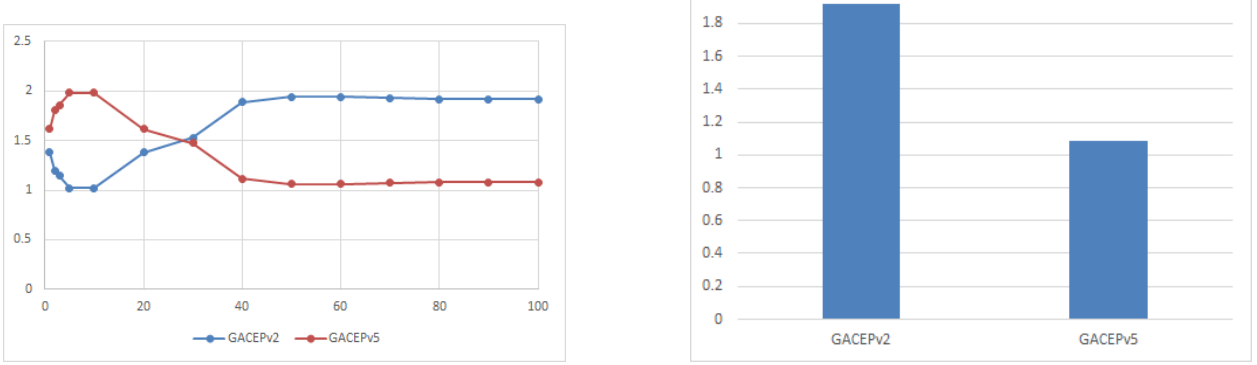
10.8.2. Versión del NAM

La motivación de esta modificación es que el cruce de dos padres lo suficientemente diferentes dará probablemente lugar a un par de soluciones que sean diferentes de ambos padres. En esta versión del NAM elegimos el primer padre mediante un Torneo Binario, para garantizar cierta bondad, y el segundo padre mediante el método original: escoger un subconjunto de la población y elegir como padre aquella solución más diferente al primer padre. Para esta modificación no hay ningún tipo de umbral de diferencias que se deba pasar para elegir al segundo padre, solo que sea

lo más distinto posible.

Esta versión tomará el nombre de **GACEPv5**.

Esta versión ya si supone una gran modificación del propio algoritmo, por lo que es necesario obtener sus resultados A.19 y compararlos con GACEPv2 para determinar si supone una mejora. Esta comparación es visible en la gráfica 10.19.



(a) Ranking durante todas las *milestones*

(b) Ranking final

Figura 10.19: Comparación de los resultados de GACEPv2 y GACEPv5

En la gráfica 10.19a se puede ver que, aunque GACEPv2 obtiene resultados mejores en prácticamente todos los casos probados, a partir del 40 % de las ejecuciones se tiene que GACEPv5 mejora a GACEPv2 en casi todas las instancias del problema probadas. Por lo tanto, podemos declarar que esta modificación supone una mejora.

10.8.3. Versión del *Crowding* Determinístico

La motivación de esta modificación es que el criterio para determinar si una nueva solución se introduce o no en la población no sea únicamente el valor de dicha solución, si no también tener en cuenta si aporta diversidad. Es decir, dada una solución nueva se deberá buscar en primer lugar aquella solución en la población con la que más parecido tenga y la sustituirá si, y solo si, la supera en valor.

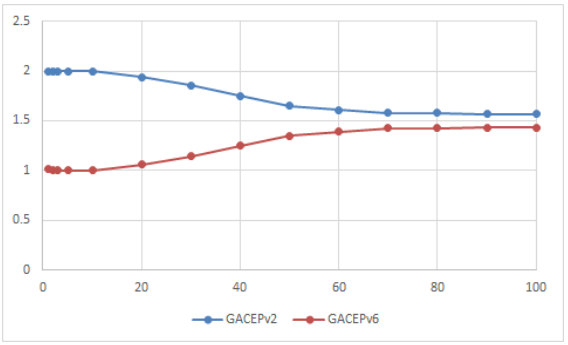
Esta versión tomará el nombre de **GACEPv6**.

De la misma forma que con la introducción del NAM, esta versión supone una gran modificación del propio algoritmo, por lo que es necesario obtener sus resultados A.20 y compararlos con GACEPv2 para determinar si se ha producido una mejora. Esta comparación es visible en la gráfica 10.20.

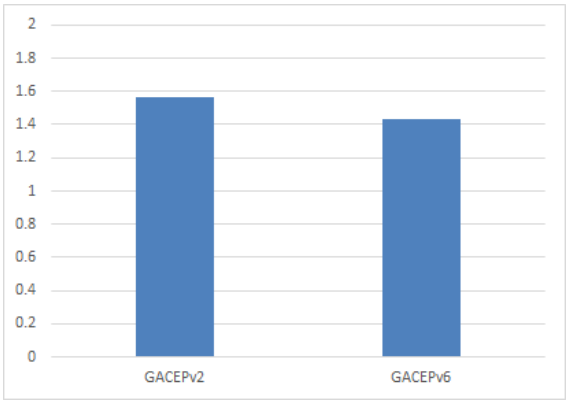
En la gráfica 10.20a se puede comprobar que la nueva versión es mejor en todas las *milestones* que GACEPv2. Sin embargo, es importante notar que solo se obtienen mejoras significantes durante la primera mitad de las iteraciones.

Reemplazo intensivo

Esta versión 30 intensifica la idea anterior, ya que no solo podrá sustituir a la solución más cercana, sino que en caso de que lo haga, se llevará un proceso de búsqueda en la población de soluciones tengan menos de cierta cantidad de diferencias con la nueva solución y nos quedaremos



(a) Ranking durante todas las *milestones*



(b) Ranking final

Figura 10.20: Comparación de los resultados de GACEPv2 y GACEPv6

únicamente con aquella que sea mejor que el resto. En tanto que el tamaño de la población debe ser constante, el resto de soluciones serán sustituidas por soluciones generadas aleatoriamente. Nótese que dicho umbral de diferencia va a ser el mismo que se ha aplicado para generar una población inicial diversa y que las soluciones generadas aleatoriamente también deberán cumplir que sean lo suficientemente diversas con respecto a la población en las que son introducidas.

Esta versión tomará el nombre de **GACEPv7**.

En tanto que es una modificación directa sobre GACEPv6, debemos comprobar primero si se consiguen mejorar los resultados sobre esta versión. Los resultados obtenidos de la ejecución de GACEPv7 se pueden encontrar en la tabla A.21 y su comparación con los resultados de GACEPv6 se puede visualizar en la gráfica 10.21 (se ha cambiado el formato de las gráficas para una mejor visualización).

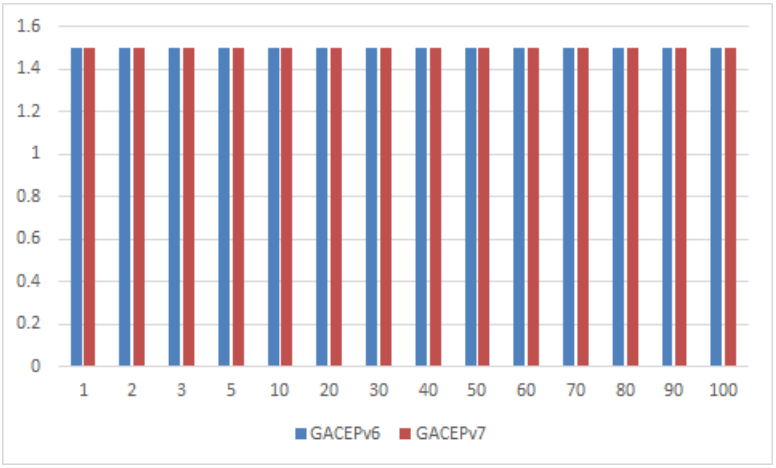


Figura 10.21: Comparación de los resultados de GACEPv6 y GACEPv7

Como se puede observar en la gráfica 10.21, ambas versiones dan exactamente los mismos resultados en todo momento. De hecho, tras obtener estos resultados se realizó un estudio de cuál era la causa y resultó que por cada hijo que se iba a introducir en la población había una o ninguna solución existente en la población dentro de este umbral, por lo que el número de soluciones a comprobar siempre era constante: una. Ya que incluso si no hay ninguna solución en dicho umbral, por la versión anterior se debe elegir aquella con mayor parecido para compararlas. Por lo tanto, esta

última modificación nunca se aplicaba. Si bien no se ha producido ninguna mejora y descartaremos esta versión, esto nos ha ayudado a descubrir que de cierta forma se está manteniendo cierta diversidad en la población, que era el objetivo de esta sección.

10.8.4. Combinaciones

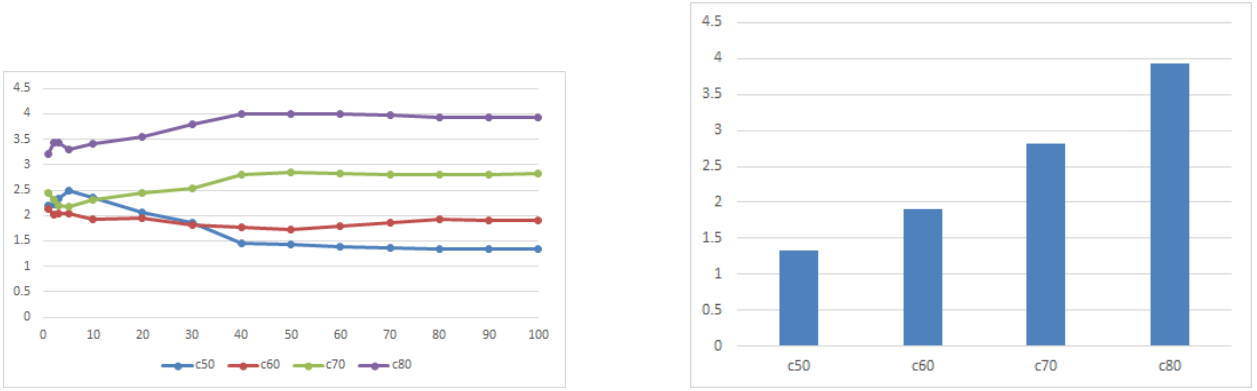
En esta sección se han obtenido resultados bastante prometedores que en todos casos mejora a su algoritmo base, GACEPv2. Por lo que ahora procederemos a analizar posibles combinaciones de dichas modificaciones por si pudiesen mejorar los resultados actuales.

GACEPv8

En primer lugar, volvemos a considerar la posibilidad de usar un operador de cruce intensivo, dado establecimos la posibilidad de que su aplicación no aportaba buenos resultados debido a la falta de diversidad. En tanto que con estas nuevas modificaciones conseguimos aumentar la diversidad, es una buena idea probar de nuevo su comportamiento junto con la adición de la versión del operador NAM. Por tanto, a la combinación de cruce intensivo y NAM se le denominará **GACEPv8**.

Los resultados de las ejecuciones de esta versión para los distintos porcentajes se encuentran en las tablas A.22, A.23, A.24 y A.25.

En la gráfica 10.22 se puede observar el comportamiento de esta nueva versión para los distintos porcentajes.



(a) Ranking durante todas las *milestones* (b) Ranking final

Figura 10.22: Comparación de los resultados de los GACEPv8cx

Se puede apreciar en 10.22a que el cruce intensivo sigue presentando las mismas características que 10.9a y 10.10a, donde los resultados son peores cuanto mayor sea la intensidad con la que se aplique este operador de cruce. Aunque también es verdad que en este caso los porcentajes 50 %, 60 % y 70 % tienen rendimiento similar durante el primer 30 % de las iteraciones y el *ranking* del 60 % se mantiene cercano al del 50 % durante toda la ejecución. Por lo que el mejor porcentaje para esta modificación sería el del 50 %.

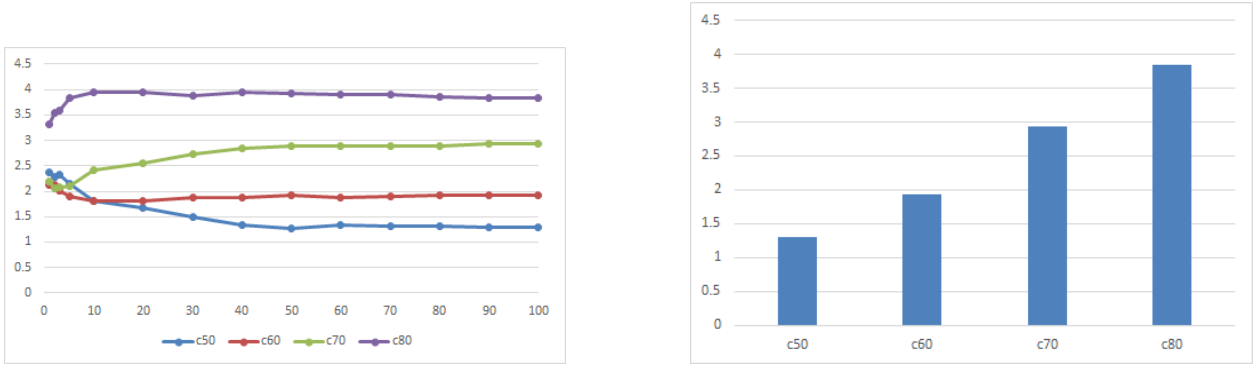
GACEPv9

A continuación, consideramos la posibilidad de usar de nuevo el operador de cruce intensivo, pero esta vez junto con la versión que implementaba la modificación del operador de reemplazo. La

motivación de esta versión es la misma que la de GACEPv8, se le quiere dar una nueva oportunidad al operador de cruce intensivo ahora que se ha aumentado la diversidad de la población en todo momento de la ejecución. Por tanto, a la combinación de cruce intensivo y *Crowding* determinístico se le denominará **GACEPv9**.

Los resultados de las ejecuciones de esta versión para los distintos porcentajes se encuentran en las tablas A.26, A.27, A.28 y A.29.

En la gráfica 10.23 se puede observar el comportamiento de esta nueva versión para los distintos porcentajes.



(a) Ranking durante todas las *milestones*

(b) Ranking final

Figura 10.23: Comparación de los resultados de los GACEPv9cx

En este caso se puede observar en 10.23a que GACEPv9 tiene un comportamiento muy similar al GACEPv8 en 10.22a en tanto que a mayor sea la intensidad del cruce peores serán los resultados y que el cruce para los porcentajes 50 % y 60 % tienen rendimientos algo más similares. En definitiva, el mejor porcentaje a utilizar para esta versión sería el del 50 %.

GACEPv10

En esta versión se quiere combinar las dos nuevas modificaciones de esta sección: la modificación del operador de selección de padres y la modificación del operador de reemplazo. La motivación de esta versión viene dada al observar las gráficas 10.19a y 10.20a. Como mencionamos respectivamente en los apartados de estas versiones, GACEPv5 consigue superar en prácticamente todos los casos a GACEPv2, pero solo a partir del 30 % de las iteraciones; mientras que GACEPv6, aunque supera a GACEPv2 para todas las *milestones*, esta mejora es solo notable hasta la primera mitad de las iteraciones. Por ello, con esta modificación se pretende combinar ambos comportamientos para que el algoritmo sea consistentemente mejor que GACEPv2. Al algoritmo resultado de combinar los operadores de NAM y *Crowding* determinístico se le denominará **GACEPv10**.

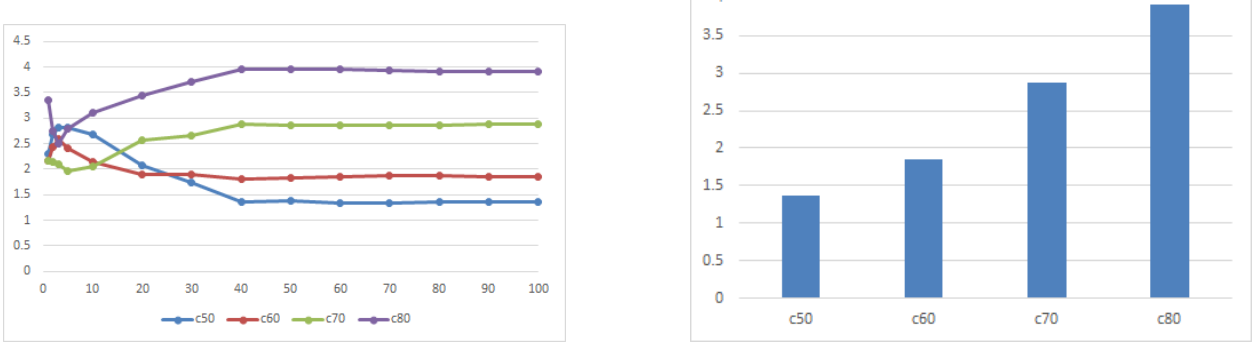
Los resultados de las ejecuciones de esta versión se encuentran en la tabla A.30.

GACEPv11

Finalmente, consideramos introducir el operador de cruce intensivo en la versión GACEPv10 basándonos en que las suposiciones sobre el comportamiento de dicha versión.La motivación de esta versión es que dicha suposición consiga mejorar el comportamiento que tiene el cruce intensivo convirtiéndolo en una versión más competente. Por tanto, al algoritmo resultante de la combinación

de cruce intensivo, operador NAM y *Crowding* determinístico se le denominará **GACEPv11**. Los resultados de las ejecuciones de esta versión para los distintos porcentajes se encuentran en las tablas A.31, A.32, A.33 y A.34.

En la gráfica 10.24 se puede observar el comportamiento de esta nueva versión para los distintos porcentajes.



(a) Ranking durante todas las *milestones*

(b) Ranking final

Figura 10.24: Comparación de los resultados de los GACEPv11cx

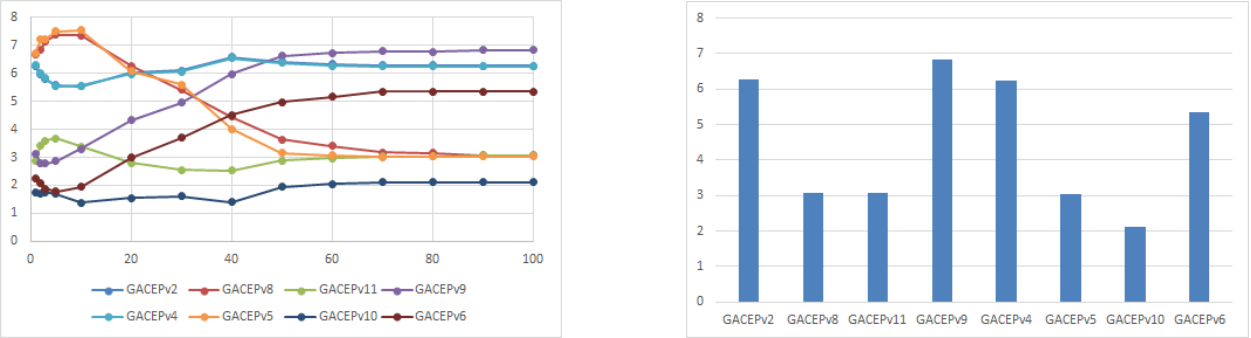
En la gráfica 10.24a se visualiza que, a grandes rasgos, se sigue manteniendo el comportamiento que lleva presentando a lo largo del trabajo, a mayor intensidad de cruce se tienen peores resultados. Además, los *ranking* del 50 % y del 60 % se mantienen parecido durante la mayoría de las iteraciones. Como novedad, se puede ver que este ha sido el único caso en el que 80 % ha tenido resultados parecidos al resto durante un breve periodo de iteraciones (en el resto de casos siempre ha sido en todo momento bastante peor).

10.8.5. Comparaciones

En este apartado se compararán todas las versiones a la vez junto con GACEPv2 para tener una visión global de los resultados que se han obtenido para esta sección y poder determinar de forma definitiva si se ha producido alguna mejora significativa y cuál versión es la mejor.

En la gráfica 10.25 vienen representado los *ranking* de cada una de las versiones para las distintas *milestones* junto con una gráfica de barras para indicar el *ranking* final de dichas versiones.

Como se puede comprobar en 10.25a todas las versiones excepto GACEPv9 ofrecen mejores resultados las instancias del problema con las que hemos trabajado. Recuérdese que GACEPv4 aportaba unos resultados muy parecidos a GACEPv2, por lo que en 10.25a se visualiza como la misma línea. Es interesante mencionar que GACEPv5 y GACEPv8 presentan un comportamiento muy similar, en las primeras iteraciones aportan los peores resultados y a medida que avanza la ejecución logra mejorar la mitad de las versiones, aunque GACEPv8 resulta ser vagamente peor. En contraposición, nos encontramos con GACEPv6 y GACEPv9, que presentan un comportamiento muy similar, en las primeras iteraciones aportan buenos resultados y a medida que avanza la ejecución obtiene peores resultados comparándolos con el resto de versiones, aunque en este caso la versión con cruce intensivo resulta bastante peor que la versión sin cruce intensivo. Por otra parte, el *ranking* de GACEPv11 se mantiene relativamente constante durante todas las iteraciones aportando uno de los mejores resultados, aunque en la segunda mitad de la ejecución su *ranking* converge con el de GACEPv5 y GACEPv8; de hecho, al final consigue el mismo *ranking* que GACEPv8, siendo



(a) Ranking durante todas las *milestones* (b) Ranking final

Figura 10.25: Comparación de los resultados de las versiones de esta sección con GACEPv2

peor que GACEPv5 por una diferencia de centésimas. Con esto llegamos a la conclusión definitiva que la introducción de cruce intensivo en el algoritmo no logra mejorarlo en ningún momento, como máximo es capaz de lograr resultados similares.

Finalmente, y más importante, hemos descubierto cuál es la mejor versión y la que consideraremos la versión final de este proceso de diseño de metaheurísticas. Se tiene que GACEPv10 es consistemente la mejor versión entre todas las presentadas en esta sección. Con eso se confirma nuestra hipótesis, presentada anteriormente, de que el uso de la versión de NAM y el cambio del operador de reemplazo iba a suponer la mezcla de los mejores comportamientos de cada versión individual: ser mejor que GACEPv2 en la mayoría de los casos durante la primera mitad de la ejecución (GACEPv6) y ser mejor que GACEPv2 en la mayoría de los casos durante la segunda mitad de la ejecución (GACEPv5).

Tras esto, mostraremos una tabla 10.12 y su gráfica 10.26 asociada sobre las diferencias entre las mejores soluciones de cada *milestone* para poder comprobar su evolución.

Tabla 10.12: Diferencias de los resultados de las distintas milestones con respecto a la final para el algoritmo GACEPv56

GACEPv10								
n	milestone	Diferencia	n	milestone	Diferencia	n	milestone	Diferencia
100	1	21.8623	**200**	1	25.5970838	**300**	1	25.4820586
	2	18.0516		2	21.1200773		2	20.90983
	3	16.1184		3	18.7431685		3	18.3898234
	5	13.1784		5	14.8269699		5	14.3143042
	10	9.95847		10	9.47807895		10	8.7815101
	20	4.83628		20	5.73939044		20	5.27891921
	30	4.23393		30	4.36629592		30	3.87182862
	40	0.13499		40	0.65991695		40	1.00562574
	50	0.08385		50	0.18139206		50	0.49856815
	60	0.01782		60	0.02886742		60	0.11276472
	70	0.01597		70	0.01139917		70	0.01587516
	80	0		80	0		80	0.00745689
	90	0		90	0		90	0

Por tanto, a la versión de GACEPv10 se le considerará la versión final, a la que nombraremos **GACEPvf**.

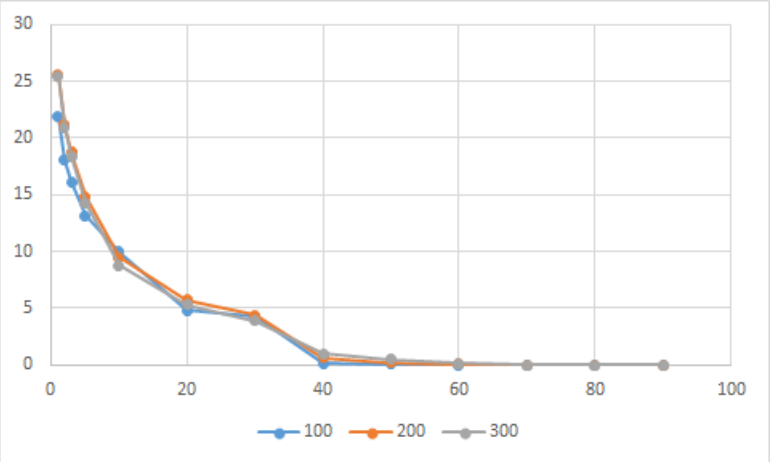


Figura 10.26: Gráfica asociada a la tabla 10.12

10.9. Comparación final

En esta sección se recordarán todas las versiones que se han utilizado a lo largo del capítulo y compararán todas las versiones más destacables (AGE, GACEPv1, GACEPv2 y GACEPv3c50) con la versión final, GACEPvf.

10.9.1. Resumen de versiones

A modo de resumen de este capítulo, reuniremos todas las versiones mencionadas en una tabla 10.13. Donde las siglas representan lo siguiente:

- **CI**: Cruce Intensivo 26.
- **PID**: Población inicial diversa 27.
- **NAM**: Versión del operador NAM 28.
- **CR**: *Crowding Replacement* 29.

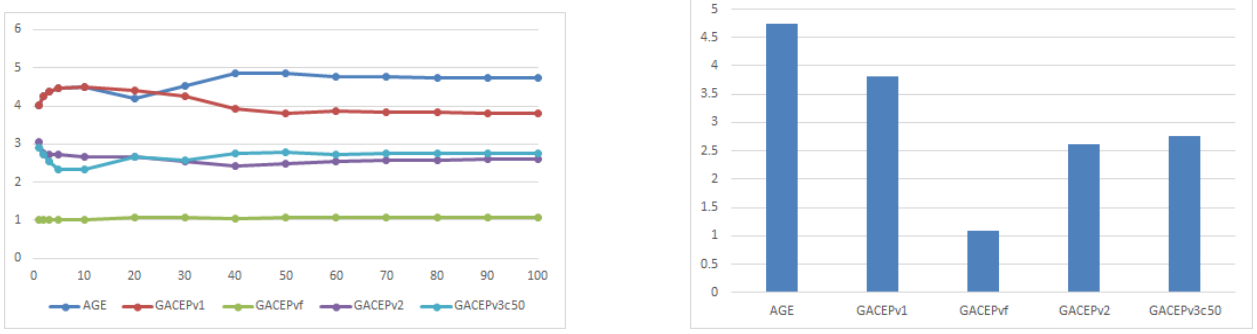
10.9.2. Comparación

En primer lugar, mostraremos una gráfica 10.27 con los *ranking* de cada una de las versiones para las distintas *milestones* junto con una gráfica de barras para indicar el *ranking* final de dichas versiones.

Además, para resaltar las diferencias entre los distintos algoritmos también se incluirá en este caso al tabla completa de los *ranking* promedios para todas la *milestones* 10.14. En dicha tabla viene representado la evolución de la clasificación de cada uno de los algoritmos en los distintos momentos de la ejecución, donde cada celda de la tabla representa el *ranking* del algoritmo que encabeza su columna con respecto al resto en la *milestone* señalada al principio de su fila. Esto se ha conseguido asignándole una clasificación a cada algoritmo por cada instancia y cada *milestone*, tras esto se calcula la media de las clasificaciones de cada algoritmo relativas todas las instancias para cada *milestone* y este valor es el que se almacenará en la tabla.

Tabla 10.13: Resumen de las distintas versiones de los algoritmos utilizadas

Algoritmo	Características
AGE	
GACEPv1	**Histórico**22
GACEPv2	Histórico, **GRASP**25
GACEPv3c	Histórico, GRASP, **CI**
GACEPv3cxwo	Histórico, **CI**
GACEPv4	Histórico, GRASP, **PID**
GACEPv5	Histórico, GRASP, PID, **NAM**
GACEPv6	Histórico, GRASP, PID, **CR**
GACEPv7	Histórico, GRASP, PID, **CRI**
GACEPv8cx	Histórico, GRASP, PID, **NAM**, **CI**
GACEPv9cx	Histórico, GRASP, PID, **CR**, **CI**
GACEPv10	Histórico, GRASP, PID, **NAM**, **CR**
GACEPv11cx	Histórico, GRASP, PID, **NAM**, **CR**, **CI**
CHC	
GACEPCHCv1	**Histórico**
GACEPCHCv2	Histórico, **GRASP**
GACEPvf	**GACEPv10**



(a) Ranking durante todas las *milestones*

(b) Ranking final

Figura 10.27: Comparación de los resultados de las versiones de esta sección con GACEPv2

En la gráfica 10.27a y en la tabla 10.14 se puede comprobar como nuestra versión final del algoritmo es mejor que el resto para prácticamente todas las instancias del problema con las que hemos trabajado en todo momento de la ejecución, ya que su *ranking* siempre es 1 o valores muy próximos a 1. Por otra parte, podemos observar en 10.27b cómo la clasificación de las distintas versiones presentadas ha ido mejorando a las anteriores. También es importante notar que para cada modificación significante se produce una gran mejora de su *ranking* (ya establecimos anteriormente que el cruce intensivo del 50 % de GACEPv3c50 es bastante similar al cruce uniforme de GACEPv2, por lo que esto no supone una modificación significativa).

Una vez que sabemos que todos estos algoritmos son mejores que el de referencia profundizaremos un poco más y estudiaremos si se ha producido una mejora significante. Esto se hará calculando la media del porcentaje de mejora de todas las soluciones de estos algoritmos con respecto a las soluciones de AGE. Es decir, para algoritmo `GACEPvx` y cada `y` *milestone* se calculará cada entrada

Tabla 10.14: Ranking de las versiones destacables para todas la milestones

	AGE	GACEPv1	GACEPvf	GACEPv2	GACEPv3c50
1	4.005155	4.005155	1.020619	3.061856	2.907216
2	4.262887	4.262887	1.010309	2.742268	2.721649
3	4.365979	4.365979	1	2.731959	2.536082
5	4.469072	4.469072	1	2.731959	2.329897
10	4.5	4.5	1	2.670103	2.329897
20	4.185567	4.412371	1.061856	2.670103	2.670103
30	4.536082	4.247423	1.072165	2.556701	2.587629
40	4.85567	3.917526	1.041237	2.43299	2.752577
50	4.845361	3.814433	1.072165	2.494845	2.773196
60	4.773196	3.876289	1.082474	2.556701	2.71134
70	4.762887	3.824742	1.082474	2.56701	2.762887
80	4.742268	3.845361	1.082474	2.587629	2.742268
90	4.742268	3.804124	1.082474	2.608247	2.762887
100	4.742268	3.804124	1.082474	2.608247	2.762887

de la tabla como:

$$\text{Mejora}_{\texttt{GACEPvx}}[\text{y}](\,\%) = \frac{\text{valor}_{\texttt{GACEPvx}}[\text{y}] - \text{valor}_{\texttt{AGE}}[\text{y}]}{\text{valor}_{\texttt{AGE}}[\text{y}]} \cdot 100 \tag{10.2}$$

donde "$\text{valor}_{\texttt{x}}[\text{y}]$" representa la media de los valores asociados a la mejores soluciones de todas las instancias del problema con el que se ha trabajado del algoritmo `x` para la *milestone* `y`. El resultado de dicha operación 10.2 será almacenado en cada celda correspondiente de la tabla 10.15. Por tanto, dicha tabla 10.15 contiene el porcentaje de mejora medio que presenta cada algoritmo para cada *milestone* con respecto a los resultados del AGE.

Tabla 10.15: Mejora porcentual media de las versiones destacables sobre AGE

	GACEPv1	GACEPv2	GACEPv3c50	GACEPv10
1	0	0.901815	1.110863221	10.652529
2	0	1.560207	1.796883169	9.5872114
3	0	1.6203	2.007854799	9.0990544
5	0	1.995523	2.55603081	8.5221226
10	0	2.581898	2.851348575	7.1178565
20	0.47747	2.30909	2.392188425	6.9379563
30	0.964412	2.715243	2.744708684	6.0968514
40	2.908935	4.972009	4.738301258	9.0957422
50	3.39837	5.308774	5.104018653	8.7712942
60	3.365162	5.369517	5.243753741	8.5328747
70	3.421865	5.390454	5.27964846	8.3044041
80	3.334575	5.249718	5.144348826	8.140204
90	3.352948	5.160873	5.057682881	8.0153536
100	3.283904	5.072261	4.969288035	7.9231946

En esta tabla 10.15 se puede comprobar que si bien GACEPv1 mejora a AGE, no parece una mejora muy significante, en tanto que no supera los resultados ni en un 3.5 % en todo momento

de la ejecución. Por otra parte, se tiene que tanto GACEPv2 y GACEPv3c50 producen mejoras significativas con respecto a los resultados originales a partir del 40 % de las iteraciones y estas mejoras se mantienen consistentes durante el resto de ejecución, siendo dichas, aproximadamente, del 5 %. Finalmente se tiene que GACEPv10 presenta mejoras significativas sobre las soluciones originales durante toda la ejecución, todas las soluciones superan a las de AGE en al menos un 6 %, siendo la media de 8.34 %.

Para confirmar que efectivamente se han producido mejoras significativas, haremos uso de un test estadístico no paramétrico para comparaciones múltiples mostrado en la Tabla 10.16.

Tabla 10.16: Test de p-valores para comparaciones múltiples referente la comparación final (Error Global: 5 %)

Algoritmos vs GACEPvf	**Original**	**Holm**	**Hommel**	**Hochberg**
AGE	1.24E-17	4.95E-17	3.25E-17	3.66E-17
GACEPv1	3.66E-17	4.95E-17	3.66E-17	3.66E-17
GACEPv2	2.16E-17	4.95E-17	3.66E-17	3.66E-17
GACEPv3c50	1.28E-17	4.95E-17	3.25E-17	3.66E-17

En 10.16 se ve de forma clara que GACEPvf mejora de forma significativa no solo el algoritmo original, sino también todas las versiones destacadas previas.

Como conclusión, procedemos a repasar las modificaciones que se han introducido en GACEPvf con el fin de analizar por qué la combinación de estas produce tan buenos resultados: Histórico, GRASP, PID, NAM y CR.

Lo que se intentó hacer en un principio fue intensificar tanto la exploración como la explotación de resultados, pero intentando siempre mantener un equilibrio entre ambas. Por ello, tras introducir el Histórico, que suponía una mejora de la explotación, en tanto que priorizaba el uso de determinadas variables que se habían supuesto buenas por pertenecer únicamente a las mejores soluciones de la población, lo lógico fue buscar una modificación que nos permitiese mejorar la exploración del espacio de soluciones sin que se superpusiese con los cambios del Histórico, en nuestro caso esto resultó en el uso del algoritmo GRASP. En tanto que ambas modificaciones nunca se utilizaban al mismo tiempo, tenemos garantizado que no iba a reducir la eficacia de forma directa de ninguno de los dos; es más, como se almacenaban los datos para usar posteriormente el histórico mientras que se usaba el algoritmo GRASP se suponía que iba a aumentar la muestra de soluciones a considerar para determinar cuáles eran mejores y cuáles peores, teniendo algo más de diversidad de elección.

Tras esto se realizó un estudio sobre la diversidad y se comprobó que era bastante insuficiente, por lo que dirigimos nuestros esfuerzos a la búsqueda de modificaciones que pudiesen incluirse en nuestro algoritmo de forma que nos permitiese aumentar la diversidad de la población. Lo mínimo se puede pedir si se quiere, como mínimo, mantener la diversidad de una población es que esta ya sea diversa; por ello, es necesario la introducción de la modificación PID. Aunque la inclusión de PID no aporte resultados novedosos, ya que sin ningún operador para mantener dicha diversidad es, a efectos prácticos, equivalente a no usar nada, esta modificación va a ser de importancia como base para los siguientes cambios.

Por otra parte, estudiamos la incorporación de las modificaciones NAM y CR por separado (10.19 y 10.20 respectivamente), obteniendo que la aplicación de NAM daba buenos resultados a partir del 40 % de las iteraciones, mientras que la aplicación de CR solo mejoraba en la mayoría de los casos hasta el 40 % de las iteraciones:

- Podemos suponer que esto ocurre ya que CR necesita de otro operador para aumentar la

diversidad, debido a que CR es un operador de reemplazo de la población, por lo que por pura probabilidad, se tiene que en algún momento las soluciones introducidas a la población sean cada vez más parecidas al resto de soluciones contenidas, perdiendo así la diversidad en etapas avanzadas de la ejecución.

- En cuanto a NAM se tiene justo lo contrario, en las primeras iteraciones obtendrá peores resultados ya al elegir como segundo padre a la solución más distinta respecto al primer padre (elegido mediante Torneo Binario, para garantizar cierta bondad) es probable que este segundo sea mucho peor, con lo que las soluciones resultantes del cruce es probable que posean un valor bastante reducido, lo que causa una disminución de reemplazos en la población y este comportamiento se mantendrá hasta que se logren introducir suficientes soluciones como mínimo decentes en la población y el uso de este operador no resulte un *handicap*.

Por lo tanto, la combinación de ambas modificaciones puede resultar en una buena idea, ya que aunque el uso de CR evita eliminar la diversidad de la población, el solo uso de este operador resulta insuficiente en tanto que no genera diversidad, por ello tenemos que combinarlo con una técnica que incorpore mayor diversidad, en nuestro caso podremos utilizar el operador NAM.

Capítulo 11: Conclusiones y trabajo futuro

El caso más habitual que se da en el diseño de algoritmos que se encuentra en la literatura implica tomar un algoritmo que es de por sí competitivo en lo relativo, al menos, al problema que se está tratando como base para el desarrollo que se va a realizar y se concluye con un análisis comparativo entre el propio algoritmo resultado del diseño y otros algoritmos competitivos de la literatura. Sin embargo, esto no ha sido posible en nuestro caso debido a la inexistencia de un algoritmo previo que tuviese como objetivo la resolución de problemas de optimización combinatorios *expensive*. Por ello, este trabajo resulta ser un estudio novedoso en el ámbito de la resolución de dichos problemas: se ha tenido que empezar desde cero, desde un algoritmo genético clásico y básico como es el AGE que no aporta especialmente buenos resultados en este ámbito de trabajo, y repasar campos de estudio de diferentes problemas con el fin de obtener inspiración para superar los inconvenientes que han surgido a lo largo del desarrollo.

Por este motivo, probablemente el mayor reto ha devenido en lidiar con los problemas propios de un proceso creativo (falta de inspiración, resultados insuficientes, ...) y en la naturaleza de un proyecto individual de tan larga duración donde la planificación y el trabajo constante han sido vitales para lograr presentar un proyecto de calidad finalizado. En este sentido, las prácticas desarrollados previamente a lo largo del doble grado (principalmente en las asignaturas del Grado de Ingeniería Informática, donde la realización de diversos trabajos para la evaluación de la parte práctica eran frecuentes), si bien de menor envergadura y mucho más guiados, fueron los cimientos para afrontar dichas situaciones adversas y superarlas adecuadamente.

Por lo tanto, podemos afirmar que este trabajo no solo destaca por presentar un algoritmo con el que se ha conseguido alcanzar satisfactoriamente el objetivo principal de este proyecto, esto es, el diseño de una metaheurística útil para ser aplicada en problemas combinatorios *expensive*, sino que lo hace también por lo siguiente:

- Se detalla cómo debe ser un proceso creativo desde el inicio cuyo objetivo sea el desarrollo de un algoritmo de esta clase.
- Se trata un problema que no ha sido explorado con anterioridad, por lo que es un estudio sin antecedentes en su campo.
- Se trata de un estudio con una gran utilidad, debido al creciente uso de problemas combinatorios en problemas complejos.

Además, ha resultado un proyecto muy completo que ha posibilitado no solo afianzar los conocimientos teóricos adquiridos durante el doble grado, sino también ampliarlos y ponerlos en práctica: obviamente ha sido necesario un predominante conocimiento sobre asignaturas orientadas al estudio de algoritmos (como sería Metaheurística y Algorítmica), se han analizado resultados usando métodos con bases matemáticas (en el ámbito de la estadística, análisis y matemáticas

aplicadas), se han consultado numerosas fuentes bibliográficas, se ha utilizado uno de los lenguajes de programación que más he usado durante estos cinco años (`C++`), se han utilizado herramientas experimentales... Como se puede comprobar, los conocimientos necesarios para llevar a cabo estas tareas provienen de una gran variedad de asignaturas del doble grado, evidenciando así la condición transversal de este proyecto.

Personalmente he disfrutado en gran medida el experimentar de primera mano cómo es el procedimiento a seguir cuando se quiere desarrollar algo nuevo y propio, aunque ha habido muchos momentos de frustración porque no lograba alcanzar los resultados deseados, todos palidecían en comparación a la alegría que se siente cuando tras tanto esfuerzo consigues superar el problema con el que te enfrentabas. También me ha parecido muy interesante todo el proceso de búsqueda de información/inspiración, ya que siempre me ha llamado la atención la cantidad de ideas tan diferentes que se le pueden ocurrir a distintas personas cuando afrontan el mismo problema; esto me permite no solo aprender más acerca de los distintos operadores que se podrían implementar, sino también me permite descubrir otros puntos de vista, aumentando mis horizontes y mejorando mi capacidad de razonamiento. Sin embargo, la parte de documentar todo el trabajo en esta memoria se me ha hecho bastante tedioso, ya que era solo formalizar cosas que ya sabía y veía muy obvias por haber estado un curso entero trabajando en ello; supongo que así es cómo se sienten muchos profesores cuando intentan explicarnos cosas que para ellos son obvias y nosotros seguimos sin enterarnos.

Para finalizar, cabe destacar que sigue siendo necesario realizar más estudios sobre cómo resolver problemas combinatorios *expensive*, dado que son problemas con una gran cantidad de aplicaciones en el mundo real, y que el algoritmo aquí presentado, como cualquier otra metaheurística existente, no está terminado, puesto que siempre queda margen de mejora. Sería interesante que se partiese del algoritmo propuesto en este trabajo y se realizasen las modificaciones sobre él y/o se utilizase para realizar comparaciones de los nuevos algoritmos que se desarrollasen (ya que al fin y al cabo es el único existente para este tipo de problemas en el momento en que se está escribiendo esto). Algunas de las modificaciones que podrían ser interesantes de comprobar serían, entre otras:

- Estudiar y aplicar otros operadores de cruce, ya que ha sido la única característica propia del AGE que se ha mantenido hasta el algoritmo final propuesto.
- Estudiar la aplicación de parámetros auto-adaptativos que no requieran que el usuario lo pre-ajuste, ya que en la literatura se ha indicado como un enfoque prometedor de desarrollo de algoritmo genéticos.

Es importante mencionar que no siempre tener un algoritmo más complejo implica que se vayan a obtener mejores resultados.

Bibliografía

[1] Q.-T. Bui, "Metaheuristic algorithms in optimizing neural network: A comparative study for forest fire susceptibility mapping in Dak Nong, Vietnam," *Geomatics, Natural Hazards and Risk*, vol. 10, n.º 1, págs. 136-150, ene. de 2019, ISSN: 1947-5705, 1947-5713. DOI: 10.1080/19475705.2018.1509902.

[2] M. B. de Moraes y G. P. Coelho, "A diversity preservation method for expensive multi-objective combinatorial optimization problems using Novel-First Tabu Search and MOEA/D," *Expert Systems with Applications*, vol. 202, pág. 117 251, sep. de 2022, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2022.117251.

[3] T. Back y H.-P. Schwefel, "Evolutionary Computation: An Overview," en *Proceedings of IEEE International Conference on Evolutionary Computation*, mayo de 1996, págs. 20-29. DOI: 10.1109/ICEC.1996.542329.

[4] S. Islam, *MATHEMATICAL PROGRAMMING*. ene. de 2020, vol. 1, pág. 700, ISBN: 978-620-0-46432-3.

[5] Z. Michalewicz, T. Baeck y D. Fogel, *Handbook of Evolutionary Computation*. Boca Raton: CRC Press, ene. de 1997, ISBN: 978-0-367-80248-6. DOI: 10.1201/9780367802486.

[6] J. Smith y T. C. Fogarty, "Operator and parameter adaptation in genetic algorithms," *Soft Computing*, vol. 1, n.º 2, ene. de 1997, ISSN: 1432-7643. DOI: 10.1007/s005000050009.

[7] Z. Tu e Y. Lu, "A Robust Stochastic Genetic Algorithm (StGA) for Global Numerical Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, n.º 5, págs. 456-470, oct. de 2004, ISSN: 1089-778X. DOI: 10.1109/TEVC.2004.831258.

[8] C. Reeves, "Genetic Algorithms," en *Handbook of Metaheuristics*, vol. 146, sep. de 2010, págs. 109-139. DOI: 10.1007/978-1-4419-1665-5_5.

[9] C. R. Reeves, "Feature Article—Genetic Algorithms for the Operations Researcher," *INFORMS Journal on Computing*, vol. 9, n.º 3, págs. 231-250, ago. de 1997, ISSN: 1091-9856. DOI: 10.1287/ijoc.9.3.231.

[10] L. J. Eshelman, "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination," en *Foundations of Genetic Algorithms*, vol. 1, Elsevier, 1991, págs. 265-283, ISBN: 978-0-08-050684-5. DOI: 10.1016/B978-0-08-050684-5.50020-3.

[11] A. Simões y E. Costa, "CHC-Based Algorithms for the Dynamic Traveling Salesman Problem," en *Applications of Evolutionary Computation*, C. Di Chio, S. Cagnoni, C. Cotta y col., eds., vol. 6624, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, págs. 354-363, ISBN: 978-3-642-20524-8 978-3-642-20525-5. DOI: 10.1007/978-3-642-20525-5_36.

[12] J.-Y. Li, Z.-H. Zhan y J. Zhang, "Evolutionary Computation for Expensive Optimization: A Survey," *Machine Intelligence Research*, vol. 19, n.º 1, págs. 3-23, feb. de 2022, ISSN: 2731-538X, 2731-5398. DOI: 10.1007/s11633-022-1317-4.

[13] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol. 9, n.º 1, págs. 3-12, ene. de 2005, ISSN: 1433-7479. DOI: 10.1007/s00500-003-0328-5.

[14] S. Shan y G. G. Wang, "Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions," *Structural and Multidisciplinary Optimization*, vol. 41, n.º 2, págs. 219-241, mar. de 2010, ISSN: 1615-1488. DOI: 10.1007/s00158-009-0420-2.

[15] Y. Tenne, C.-K. Goh, L. M. Hiot e Y. S. Ong, eds., *Computational Intelligence in Expensive Optimization Problems* (Adaptation Learning and Optimization). Berlin, Heidelberg: Springer, 2010, vol. 2, ISBN: 978-3-642-10700-9 978-3-642-10701-6. DOI: 10.1007/978-3-642-10701-6.

[16] A. D. Martinez, J. Del Ser, E. Villar-Rodriguez, E. Osaba, J. Poyatos, S. Tabik, D. Molina y F. Herrera, "Lights and shadows in Evolutionary Deep Learning: Taxonomy, critical methodological analysis, cases of study, learned lessons, recommendations and challenges," *Information Fusion*, vol. 67, págs. 161-194, mar. de 2021, ISSN: 1566-2535. DOI: 10.1016/j.inffus.2020.10.014.

[17] J. Poyatos, D. Molina, A. D. Martinez, J. Del Ser y F. Herrera, "EvoPruneDeepTL: An evolutionary pruning model for transfer learning based deep neural networks," *Neural Networks*, vol. 158, págs. 59-82, ene. de 2023, ISSN: 0893-6080. DOI: 10.1016/j.neunet.2022.10.011.

[18] E. Pellerin, L. Pigeon y S. Delisle, "Self-adaptive parameters in genetic algorithms," en *Defense and Security*, B. V. Dasarathy, ed., Orlando, FL, abr. de 2004, pág. 53. DOI: 10.1117/12.542156.

[19] A. Eiben, R. Hinterding y Z. Michalewicz, "Parameter Control in Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 3, n.º 2, págs. 124-141, jul. de 1999, ISSN: 1941-0026. DOI: 10.1109/4235.771166.

[20] F. Lobo y D. Goldberg, "An Overview of the Parameter-Less Genetic Algorithm," ene. de 2008.

[21] J. Lis, "Parallel Genetic Algorithm with the Dynamic Control Parameter," en *Proceedings of IEEE International Conference on Evolutionary Computation*, mayo de 1996, págs. 324-329. DOI: 10.1109/ICEC.1996.542383.

[22] J. J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, n.º 1, págs. 122-128, ene. de 1986, ISSN: 2168-2909. DOI: 10.1109/TSMC.1986.289288.

[23] T. Pham, *Competitive Evolution: A Natural Approach to Operator Selection.* ene. de 1994, vol. 956, pág. 60, ISBN: 978-3-540-60154-8. DOI: 10.1007/3-540-60154-6_47.

[24] D. Wolpert y W. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, n.º 1, págs. 67-82, abr. de 1997, ISSN: 1941-0026. DOI: 10.1109/4235.585893.

[25] S. Chaturvedi y V. P. Sharma, "A Modified Genetic Algorithm Based on Improved Crossover Array Approach," en *Advances in Data and Information Sciences*, M. L. Kolhe, M. C. Trivedi, S. Tiwari y V. K. Singh, eds., ép. Lecture Notes in Networks and Systems, Singapore: Springer, 2019, págs. 117-127, ISBN: 9789811302770. DOI: 10.1007/978-981-13-0277-0_10.

[26] R. Cheng y M. Gen, "Crossover on intensive search and traveling salesman problem," *Computers & Industrial Engineering*, 16th Annual Conference on Computers and Industrial Engineering, vol. 27, n.º 1, págs. 485-488, sep. de 1994, ISSN: 0360-8352. DOI: 10.1016/0360-8352(94)90340-9.

[27] Z. Qiongbing y D. Lixin, "A new crossover mechanism for genetic algorithms with variable-length chromosomes for path optimization problems," *Expert Systems with Applications*, vol. 60, págs. 183-189, oct. de 2016, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2016.04.005.

[28] C. Fernandes, A. Rosa, A. Pais y T. Norte, "A Study on Non-random Mating and Varying Population Size in Genetic Algorithms Using a Royal Road Function," abr. de 2001.

[29] D. E. Goldberg y K. Deb, "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms," en *Foundations of Genetic Algorithms*, vol. 1, Elsevier, 1991, págs. 69-93, ISBN: 978-0-08-050684-5. DOI: 10.1016/B978-0-08-050684-5.50008-2.

[30] S. W. Mahfoud, "Crowding and Preselection Revisited," en *Parallel Problem Solving from Nature*, 1992.

[31] G. Gallo, P. L. Hammer y B. Simeone, "Quadratic knapsack problems," en *Combinatorial Optimization*, ép. Mathematical Programming Studies, M. W. Padberg, ed., Berlin, Heidelberg: Springer, 1980, págs. 132-149, ISBN: 978-3-642-00802-3. DOI: 10.1007/BFb0120892.

[32] C. Witzgall, "Mathematical Methods of Site Selection for Electronic Message Systems (EMS)," *NASA STI/Recon Technical Report N*, vol. 76, pág. 18 321, jun. de 1975.

[33] J. M. W. Rhys, "A Selection Problem of Shared Fixed Costs and Network Flows," *Management Science*, vol. 17, n.º 3, págs. 200-207, nov. de 1970, ISSN: 0025-1909, 1526-5501. DOI: 10.1287/mnsc.17.3.200.

[34] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel y L. A. Wolsey, "Formulations and valid inequalities for the node capacitated graph partitioning problem," *Mathematical Programming*, vol. 74, n.º 3, págs. 247-266, sep. de 1996, ISSN: 1436-4646. DOI: 10.1007/BF02592198.

[35] D. Pisinger, "The quadratic knapsack problem—a survey," *Discrete Applied Mathematics*, vol. 155, n.º 5, págs. 623-648, mar. de 2007, ISSN: 0166218X. DOI: 10.1016/j.dam.2006.08.007.

[36] A. LaTorre, D. Molina, E. Osaba, J. Poyatos, J. Del Ser y F. Herrera, "A prescription of methodological guidelines for comparing bio-inspired optimization algorithms," *Swarm and Evolutionary Computation*, vol. 67, pág. 100 973, dic. de 2021, ISSN: 2210-6502. DOI: 10.1016/j.swevo.2021.100973.

[37] J. Derrac, S. García, D. Molina y F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, n.º 1, págs. 3-18, mar. de 2011, ISSN: 22106502. DOI: 10.1016/j.swevo.2011.02.002.

[38] S. García, D. Molina, M. Lozano y F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 Special Session on Real Parameter Optimization," *Journal of Heuristics*, vol. 15, n.º 6, págs. 617-644, dic. de 2009, ISSN: 1572-9397. DOI: 10.1007/s10732-008-9080-4.

[39] D. Rader Jr y G. Woeginger, "The Quadratic 0–1 Knapsack Problem with Series–Parallel Support," *Operations Research Letters*, vol. 30, págs. 159-166, jun. de 2002. DOI: 10.1016/S0167-6377(02)00122-0.

[40] A. Herrera-Poyatos y F. Herrera, *Genetic and Memetic Algorithm with Diversity Equilibrium based on Greedy Diversification*, feb. de 2017. arXiv: 1702.03594 [cs].

[41] C. García-Martínez, F. Glover, F. J. Rodriguez, M. Lozano y R. Martí, "Strategic oscillation for the quadratic multiple knapsack problem," *Computational Optimization and Applications*, vol. 58, n.º 1, págs. 161-185, mayo de 2014, ISSN: 0926-6003, 1573-2894. DOI: 10.1007/s10589-013-9623-y.

[42] F. Oppacher y M. Wineberg, "The Shifting Balance Genetic Algorithm: Improving the GA in a Dynamic Environment," en *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*, ép. GECCO'99, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., jul. de 1999, págs. 504-510, ISBN: 978-1-55860-611-1.

Apéndice A: Apéndice: Tablas de Resultados Individuales

En este apéndice se incluyen los resultados individuales de cada algoritmo, para quienes les interese conocerlos, que se han presentado y detallado en el capítulo 10.

También se vuelve a mencionar que todos estos resultados y más que no se han acabado incluyendo en la memoria se pueden encontrar en el respositorio de Github de este trabajo[1]; donde se encuentran los siguientes ficheros:

- Los ficheros de texto resultantes de la ejecución de cada algoritmo.

- El equivalente de estos ficheros anteriores en formato de excel con todas las tablas y gráficas usadas.

- Otros ficheros de excel que se han utilizado para realizar las comparaciones entre distintos algoritmos con el formato adecuado para introducir dichos archivos directamente en Tacolab[2] (el cual hemos utilizado para calcular los *ranking* y los distintos tests estadísticos no paramétricos)

- Ficheros de excel que contienen las tablas de los distintos *rankings* junto con las gráficas asociadas.

[1] https://github.com/Itrilor/TFG/tree/main/Subresultados/50tries
[2] https://tacolab.org/comp

Tabla A.1: Resultados de la ejecución de AGEU con 450 iteraciones

AGEU 450								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	192975	**200**	25	375031	**300**	25	376494
		81641.3			936531			28788.7
		189162			302452			275573
		224189			29203.7			439862
		229585			100514			14777.8
		73991.5			779113			263938
		10154.9			40194.9			481905
		62289.6			693305			9176.36
		232572			779359			245377
		24729.5		50	622413		50	990137
	50	18382.8			47981			501953
		56493.1			203099			104754
		3703.2			239245			866729
		50077.8			220724			301858
		61451.4			185943			716621
		35971.6			79424.5			722726
		14528.2			58364.2			43401.7
		20289.4			147914			756135
		35073.3			48982.8			751915
		88195.5		75	281677			
	75	83363			369024			
		104674			209377			
		33684.2			225289			
		105760			225869			
		56123.2			479039			
		16040.6			425768			
		52535.1			218123			
		53964			315253			
		68468.1			104110			
		143555		100	142039			
	100	188657			439482			
		94679.5			283241			
		61561.5			61605.9			
		71896.1			127029			
		27565.6			137458			
		144945			228081			
		110542			267388			
		19419			596574			
		103743			512266			

Tabla A.2: Resultados de la ejecución de CHC con 450 iteraciones

CHC 450								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	191858	**200**	25	372331	**300**	25	371526
		81329.3			934965			28637.6
		188291			299738			272165
		223260			28848.1			434626
		228377			99231.6			14864.6
		73621.4			774945			259391
		10099.3			39613.1			477525
		61818.1			689720			9258.1
		231871			773943			240501
	50	24609.9		50	619791		50	984358
		18473.4			47923.5			493627
		55576.8			200623			102690
		3700.86			237936			858793
		49625.6			219477			297041
		60877.7			184164			710705
		35773.1			78748.7			716775
		14529.6			58104.2			43010.5
		20267.8			146223			748124
		34826.2			48936.9			746153
	75	87613.6		75	279742			
		82863.9			365710			
		103618			207919			
		33720.1			222401			
		104278			222974			
		55744.3			477196			
		16033.5			423271			
		52231.4			216051			
		53512.8			312565			
		67837.3			103343			
	100	142030		100	140595			
		188444			435501			
		94148.3			280027			
		61016.8			61136.7			
		71666.1			125267			
		27565.1			136405			
		143694			225762			
		109763			264457			
		19448.6			592833			
		103480			508497			

Tabla A.3: Resultados de la ejecución de GACEPv1 (solo mutación)

GACEPv1 (solo Mutación)								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	192911	**200**	25	375846	**300**	25	376538
		81586			936491			28755
		189000			302491			276365
		224165			29209			439571
		229612			100396			14783.2
		74134.8			778754			265167
		10150			40160.4			482070
		62350.1			692661			9193.24
		232576			779092			245600
		24733.8		50	622410		50	990758
	50	18428.9			48034.3			503530
		56497.6			203104			104690
		3702.9			239163			866804
		50095.3			220889			303417
		61451.8			186269			716685
		35980.1			79650.4			723594
		14516			58451.4			43392.5
		20282.4			148118			757199
		35094.8			49022.3			751429
		88213.6		75	281832			
	75	83410.4			368948			
		104665			209455			
		33708.5			225696			
		105736			226533			
		56226.8			479074			
		16040.9			425712			
		52615.7			218742			
		54068.7			315513			
		68395			104141			
		143406		100	142057			
	100	188657			439810			
		94670.7			283234			
		61873.9			61647			
		71899.6			127426			
		27569.9			137529			
		144827			227890			
		110432			267261			
		19433.4			596528			
		103778			512408			

Tabla A.4: Resultados de la ejecución de GACEPv1 (solo cruce)

GACEPv1 (solo Cruce)								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	203301	**200**	25	370871	**300**	25	372493
		85875			961415			30473.8
		203294			309892			260902
		234769			32099.8			440379
		238204			107039			15728.8
		79624.8			807116			244485
		10630.9			43845			489256
		66357.6			712826			9816.72
		242913			809535			225420
		26129.9		50	638763		50	1009440
	50	20105.3			48925.2			463674
		57731.3			207052			105597
		3866.06			242659			875417
		52713.3			226599			294907
		62639.1			190105			709720
		37946.5			77773.4			723854
		15568.9			59112.9			46575.9
		21663.3			146561			751944
		36519.9			50104.3			752260
		93768.5		75	283640			
	75	88446.5			378210			
		108413			205895			
		36258.3			217994			
		109777			219976			
		59204.5			490868			
		17550.7			439293			
		54727.4			214437			
		57408.9			319593			
		72713.6			107720			
		151748		100	147471			
	100	191674			445096			
		101806			273655			
		65318.5			66192.4			
		78578.8			131752			
		30879.6			144761			
		150127			231237			
		118070			266288			
		22066.2			615510			
		111587			526593			

Tabla A.5: Resultados de la ejecución de GACEPv1

GACEPv1								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	203917	**200**	25	371906	**300**	25	370202
		86893.8			960695			30668.8
		202056			310368			261697
		234528			31956			441509
		238931			107874			15628.8
		78698.4			804278			245342
		10611.2			43930.7			487566
		67007.1			711466			9749.8
		241977			805644			226145
		26107.5		50	639675		50	1009100
	50	20195.7			49255.1			469319
		58108.8			206394			105789
		3841.06			243532			873907
		52534.5			226204			295377
		62661.5			189500			706744
		38120.5			78811.9			719059
		15454.7			59037			46737.7
		21423.6			146471			751434
		36782.8			49706.3			752530
		94302.4		75	283170			
	75	88189.9			374933			
		108332			205342			
		36961.9			219207			
		110489			220104			
		57749			490903			
		17239.7			439020			
		55209.6			214588			
		57800.7			318589			
		72183.1			108423			
		151073		100	148614			
	100	191744			445441			
		101933			277919			
		65749.4			65806.9			
		78826.7			133560			
		30323.9			144762			
		150580			232794			
		118139			269788			
		21744.5			616128			
		111748			525471			

Tabla A.6: Resultados de la ejecución de GACEPv2

GACEPv2								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	203990	**200**	25	387130	**300**	25	382607
		86847.5			957795			30884.8
		202881			316415			268992
		235412			31924.3			451159
		239531			106308			15802.6
		81629.6			815068			254954
		11202.5			43813.2			495206
		67541			730170			9573.32
		240632			812688			235222
		26650		50	654376		50	1018740
	50	20339.9			50821.3			481431
		58377.6			210995			110335
		3865.52			244355			897264
		52691.9			228102			309758
		61917.1			193566			722396
		38957.8			82207.8			739186
		15741.1			61324.7			46235.8
		21734.1			150770			770360
		37536			51536.6			766037
		94164.3		75	288806			
	75	89106.7			384827			
		110585			211528			
		37603.8			227305			
		109186			227240			
		60335			489482			
		18012.9			440688			
		56660.6			221885			
		58571.4			325370			
		74388.2			111179			
	100	150944		100	152047			
		191323			455399			
		104908			286076			
		67064.8			65802			
		79579.5			136201			
		30680.3			148119			
		153433			237716			
		119628			277706			
		21846.9			620681			
		113565			534615			

Tabla A.7: Resultados de la ejecución de GACEPv3c50

GACEPv3c50								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	203757	**200**	25	387692	**300**	25	380873
		87986.7			960604			31037.8
		201925			319451			270676
		234687			31800.2			453817
		239168			106114			15834.7
		80762.7			811815			256274
		10804.4			43245.2			495113
		67338.3			724869			9635.3
		239113			813379			240383
		26613.5		50	648464		50	1017950
	50	20342.2			50783.1			484502
		57992.1			209954			110387
		3924.16			244967			892127
		52512.4			227277			312468
		61926.7			193021			724628
		38691			82316.6			739209
		15665.2			61604.7			46201.5
		21979.3			150622			771481
		37590.6			51886.7			771581
		94105.8		75	288139			
	75	88436.9			383285			
		110483			214331			
		37267.6			228410			
		109713			228443			
		60655.5			488552			
		17397			439836			
		56051.5			221324			
		58545.3			325709			
		73755.1			111450			
		151126		100	151553			
	100	191042			451736			
		104092			287180			
		66607.5			66302.7			
		79544.2			135140			
		30642.7			147292			
		153030			240391			
		120108			275914			
		21438.8			621167			
		113758			538408			

Tabla A.8: Resultados de la ejecución de GACEPv3c60

GACEPv3c60								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	203538	**200**	25	384933	**300**	25	378030
		87503.7			955992			30762.9
		200486			315308			268113
		233820			31962.3			451102
		236577			106034			15863.5
		80107.2			811910			257014
		10766.7			43073.9			492894
		66426.1			722385			9562.72
		238292			809614			239399
		26644.5		50	647707		50	1015430
	50	20138.6			50057.6			492598
		57833.4			210341			110034
		3852.42			243681			888694
		52395.7			226608			310748
		61856.5			192628			718529
		38257.8			82363.8			735245
		15583.5			61443.6			45922.7
		21609.3			148549			768903
		36991.5			51237.3			758744
		93467		75	285767			
	75	88365.5			381106			
		110012			210445			
		36997.7			225648			
		108988			225940			
		59982.4			485967			
		17449.5			438845			
		55740.1			220748			
		58134.1			322486			
		73036.6			110494			
		149657		100	149769			
	100	191062			453784			
		102578			283688			
		66332.3			65864.3			
		78653.5			135016			
		30318.1			146384			
		151617			238059			
		117572			273827			
		21576.7			617805			
		112050			532984			

Tabla A.9: Resultados de la ejecución de GACEPv3c70

GACEPv3c70								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	202531	**200**	25	388242	**300**	25	372547
		85814.5			953518			30522
		199018			314018			265953
		231465			30947.1			446872
		235265			105444			15541.5
		79043.2			804681			256101
		10791.3			42646.5			491823
		66706.7			714956			9566.16
		239021			802000			240818
		26055.6		50	638680		50	1010440
	50	19894			49853.1			489849
		57814.6			208449			108718
		3878.42			243525			881565
		51888			226116			309830
		61870.2			191190			715122
		37885.9			81224.7			729050
		15403			60392.9			45411.8
		21567.9			147551			764464
		36429.7			51049			757350
		92926		75	283107			
	75	88050.8			377660			
		108643			210677			
		36145.8			226423			
		108796			224484			
		58956.1			487463			
		17141.8			435510			
		54622			218490			
		56990.9			318913			
		72073.7			109300			
		149191		100	148718			
	100	190861			444767			
		101231			282298			
		65845.2			65654.4			
		77769.4			133901			
		29711.6			145012			
		151575			237391			
		116671			272921			
		21203.6			615828			
		111391			525722			

Tabla A.10: Resultados de la ejecución de GACEPv3c80

GACEPv3c80								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	200758	**200**	25	383378	**300**	25	369966
		85001.1			951808			29779.4
		197164			311215			268528
		230385			30825.4			445004
		234759			103925			15323.3
		77138.7			796957			259133
		10508.3			42431.8			486831
		65090.9			705864			9483.96
		237575			797724			242920
		25511.7		50	628684		50	1004060
	50	19375.6			49241.6			496973
		57494.6			206083			106882
		3830.34			242641			878279
		51415.5			224679			305530
		61972.8			188763			702466
		37465.9			80695.3			712944
		15201.1			59879.2			45209.2
		21197.9			145128			753796
		36108.9			50305.8			745203
		91488.2		75	278414			
	75	86406.6			374658			
		108313			209105			
		35623			224626			
		108130			225864			
		58410.6			485996			
		17048.6			434028			
		53956.9			217732			
		56466.5			315429			
		70869.1			107448			
		147471		100	146360			
	100	190711			436927			
		99373.5			282575			
		64308.7			64700.6			
		75211			130894			
		29448.9			142502			
		148569			232987			
		115049			271102			
		20559.4			609045			
		108835			517959			

Tabla A.11: Resultados de la ejecución de GACEPv3c50wo

GACEPv3c50wo								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	204252	**200**	25	373301	**300**	25	369510
		86315.5			959578			30646.8
		202935			309898			260537
		235175			32002.9			440466
		237144			106823			15625
		80045			806153			247980
		10685.6			43339			487948
		67601.2			710409			9608.7
		241377			805424			230013
		25724.7		50	638533		50	1009700
	50	19918.8			49389.8			466416
		57718.8			207116			105911
		3880.5			242587			873005
		52696.2			225620			296467
		62702.2			189404			707170
		37626			78689.9			721520
		15323.6			59118.7			46674.9
		21467.5			146394			748156
		36660.6			50088.1			749926
		93471.2		75	280806			
	75	87771.7			375619			
		108569			203080			
		36392			221878			
		110087			222739			
		59007.6			490520			
		17413.9			438736			
		55186.4			213253			
		56642.6			317841			
		73337.8			108347			
		150406		100	148570			
	100	191183			444807			
		102565			277386			
		65861			66315.8			
		77558.4			131875			
		31105.8			145072			
		151274			232040			
		117649			272055			
		21727			612031			
		111113			525041			

Tabla A.12: Resultados de la ejecución de GACEPv3c60wo

GACEPv3c60wo								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	203749	**200**	25	372464	**300**	25	369511
		85335			957945			30417.1
		201355			305642			260412
		233048			31560.4			437943
		236343			106797			15630.3
		78427.5			801631			247298
		10523			43245.9			486597
		66553.4			709005			9629.22
		241741			800107			226616
	50	25803.5		50	636635		50	1003660
		19707.5			49315.5			466449
		57904.6			205776			105546
		3866.32			242907			871239
		52377.6			224955			294255
		62848.3			188539			700755
		37592.7			79707.2			712866
		15435.1			59223.7			45991.6
		21240.2			144671			747411
		36544.5			49948.4			742119
	75	92731.6		75	281988			
		87380.2			373362			
		108548			205536			
		36191.7			222249			
		109292			224278			
		58746.1			489324			
		17041.5			437577			
		54638.1			213068			
		56662.6			315208			
		71005.3			108306			
	100	150196		100	146054			
		191534			437187			
		101253			278468			
		65025.3			65276.8			
		77326.4			130703			
		30204.3			142745			
		150939			232323			
		116270			268344			
		21484.8			611407			
		108996			521768			

Tabla A.13: Resultados de la ejecución de GACEPv3c70wo

GACEPv3c70wo								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	200773	**200**	25	370832	**300**	25	362818
		84348.6			956117			30136.3
		197399			305785			259223
		231469			31102.6			434318
		235318			105627			15482.1
		77922.3			794401			246581
		10501			42443			483927
		65821			705677			9690.54
		239530			794817			232418
	50	25442.8		50	626258		50	1001240
		19457.7			49041.6			472349
		57610.4			203770			105973
		3842.98			242502			864944
		51758.4			223724			297463
		62479.4			187465			695229
		37155			79020.5			707297
		15112.9			59128.2			45371.3
		21039.7			144627			739062
		36066.7			49425.9			737015
	75	92244.2		75	277162			
		87080.3			370614			
		108286			204126			
		35776.2			221738			
		109086			223024			
		57922.2			487477			
		17101.4			433978			
		54030.1			211357			
		56268.4			312605			
		70436.8			106169			
	100	149628		100	146203			
		191113			433820			
		100325			273505			
		64331.4			64828.6			
		75988.4			130238			
		29895			142201			
		149885			230640			
		114765			266802			
		21157.4			607605			
		108482			513407			

Tabla A.14: Resultados de la ejecución de GACEPv3c80wo

GACEPv3c80wo								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	199168	**200**	25	371648	**300**	25	360258
		84602			953777			29505.6
		195825			304553			258750
		230702			30725.4			433603
		233001			103604			15310.1
		76610.1			784164			248562
		10437			42597.6			481412
		64850.2			691512			9586.68
		239689			787417			233770
		25080		50	621036		50	993622
	50	19177.2			48184			475483
		57454.1			203380			105026
		3844.7			242407			860904
		51448.6			222863			297678
		62248.7			185272			688284
		36643			78588.2			699460
		15268.3			57966.1			44865.8
		20909.9			143240			732157
		35312.6			48840.1			730693
		91027.4		75	273375			
	75	85530.6			366781			
		108027			206021			
		35462.6			221356			
		108097			220509			
		56920.3			485328			
		16898.1			431343			
		53409.1			211606			
		55948.5			309322			
		69226.3			105205			
		146513		100	142518			
	100	190902			427445			
		98183.7			275612			
		63294.7			64695			
		74382.1			127100			
		29589.8			139809			
		148304			228733			
		113298			266189			
		20939.4			602926			
		106343			508022			

Tabla A.15: Resultados de la ejecución del GACEPCHCv1

GACEPCHCv1								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	194639	**200**	25	373591	**300**	25	369410
		86869			933109			27782.8
		196923			294904			270387
		222496			32013			432954
		228452			100539			14193.9
		82049.3			780958			253646
		12385.9			44160.1			474080
		68039.3			698695			9466.94
		231297			778611			234882
		26731.7		50	643395		50	976676
	50	20914.2			47924.2			483925
		54578.7			198093			98400.5
		4641.04			236354			858215
		52339.2			216292			286112
		60366.2			184896			708915
		38772			80298.5			714617
		16132.9			58804.4			42775.4
		22053.8			147361			742856
		37824.3			48560.2			740692
		94260		75	285114			
	75	88509.2			367262			
		104608			212544			
		36897.1			225521			
		103568			225152			
		61144.7			475673			
		18779.3			420033			
		57608.6			219742			
		58576.6			319014			
		73902.4			102055			
		145867		100	142647			
	100	188548			446476			
		105956			287128			
		69068.9			61039.8			
		80575.2			130393			
		30618.2			138659			
		147531			228818			
		122807			270042			
		23740.4			593472			
		114929			516373			

Tabla A.16: Resultados de la ejecución de GACEPCHCv2

Diversidad en GACEPv2								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	0.00713778	**200**	25	0.0157289	**300**	25	0.0112578
		0.00725778			0.00213556			0.00259852
		0.00654222			0.0122222			0.0138519
		0.00174667			0.00138			0.00867407
		0.0028			0.00182444			0.0014163
		0.0123067			0.00575556			0.0114356
		0			0.000346667			0.00479704
		0.01072			0.00811111			0.000906667
		0.00251556			0.00475333			0.013
		0.0129289		50	0.00761333		50	0.00608
	50	0.00721333			0.00785778			0.0132859
		0.00473778			0.00976444			0.00506222
		0.000875556			0.00113111			0.00912148
		0.00622222			0.00181556			0.0123748
		0.00127556			0.00587778			0.0083837
		0.0113511			0.0126933			0.0115378
		0.00651111			0.0139644			0.000791111
		0.00341778			0.0126244			0.0059437
		0.0122844			0.00907111			0.0105378
		0.00459111		75	0.0136533			
	75	0.00806222			0.00640889			
		0.00485333			0.0133556			
		0.00296444			0.0148622			
		0.00395556			0.0115089			
		0.0133911			0.000522222			
		0.00437778			0.00457778			
		0.0165644			0.00859111			
		0.0160667			0.0146356			
		0.00919111			0.0149067			
		0.00152		100	0.00532444			
	100	0.00008			0.0144578			
		0.0141778			0.01566			
		0.0115822			0.00266			
		0.0163911			0.00588			
		0.00313333			0.00626667			
		0.00872444			0.00773778			
		0.00598667			0.0128267			
		0.00354667			0.00678222			
		0.0109956			0.00716			

Tabla A.17: Diversidad de la población de GACEPCHCv2

Diversidad en GACEPv2								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	0.00713778	**200**	25	0.0157289	**300**	25	0.0112578
		0.00725778			0.00213556			0.00259852
		0.00654222			0.0122222			0.0138519
		0.00174667			0.00138			0.00867407
		0.0028			0.00182444			0.0014163
		0.0123067			0.00575556			0.0114356
		0			0.000346667			0.00479704
		0.01072			0.00811111			0.000906667
		0.00251556			0.00475333			0.013
		0.0129289		50	0.00761333		50	0.00608
	50	0.00721333			0.00785778			0.0132859
		0.00473778			0.00976444			0.00506222
		0.000875556			0.00113111			0.00912148
		0.00622222			0.00181556			0.0123748
		0.00127556			0.00587778			0.0083837
		0.0113511			0.0126933			0.0115378
		0.00651111			0.0139644			0.000791111
		0.00341778			0.0126244			0.0059437
		0.0122844			0.00907111			0.0105378
		0.00459111		75	0.0136533			
	75	0.00806222			0.00640889			
		0.00485333			0.0133556			
		0.00296444			0.0148622			
		0.00395556			0.0115089			
		0.0133911			0.000522222			
		0.00437778			0.00457778			
		0.0165644			0.00859111			
		0.0160667			0.0146356			
		0.00919111			0.0149067			
	100	0.00152		100	0.00532444			
		0.00008			0.0144578			
		0.0141778			0.01566			
		0.0115822			0.00266			
		0.0163911			0.00588			
		0.00313333			0.00626667			
		0.00872444			0.00773778			
		0.00598667			0.0128267			
		0.00354667			0.00678222			
		0.0109956			0.00716			

Tabla A.18: Resultados de la ejecución de GACEPv4

GACEPv4								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	203990	**200**	25	387130	**300**	25	382607
		86847.5			957795			30884.8
		202881			316415			268992
		235412			31924.3			451159
		239531			106308			15787.6
		81629.6			815068			254954
		11202.5			43804			495206
		67541			730170			9720.7
		240632			812688			235222
		26650		50	654376		50	1018740
	50	20339.9			50821.3			481431
		58377.6			210995			110335
		3865.52			244355			897264
		52691.9			228102			309758
		61919.9			193566			722396
		38957.8			82207.8			739186
		15741.1			61324.7			46235.8
		21734.1			150770			770360
		37536			51536.6			766037
		94164.3		75	288806			
	75	89106.7			384827			
		110585			211528			
		37603.8			227305			
		109186			227240			
		60335			489482			
		18012.9			440688			
		56660.6			221885			
		58571.4			325370			
		74388.2			111179			
		150944		100	152047			
	100	191683			455399			
		104908			286076			
		67064.8			65802			
		79579.5			136201			
		30680.3			148119			
		153433			237716			
		119628			277706			
		21846.9			620681			
		113565			534615			

Tabla A.19: Resultados de la ejecución de GACEPv5

GACEPv5								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	208754	**200**	25	391926	**300**	25	387039
		91678.1			968126			31634.5
		208787			314694			274231
		240594			33597.8			454235
		244078			110667			16458.2
		84602.7			826138			254241
		11969.9			44976.3			499108
		70101.1			735609			10085.2
		244600			826376			238168
		27444.9		50	665063		50	1030220
	50	20823			50739.5			489681
		59247.9			213414			111870
		4099.78			246819			900266
		53997.1			230038			308827
		62641.1			195599			734700
		39338			82476.5			751495
		16505.2			61497.9			48133.9
		22959.3			152281			780263
		38224.4			52357.9			776609
		96287.5		75	294494			
	75	90981.6			387527			
		113115			214621			
		38346.1			229274			
		111103			226564			
		61713.6			492729			
		19185.6			445501			
		57782.1			224217			
		60335.8			329457			
		76176.1			111040			
		154076		100	152759			
	100	191824			462556			
		107298			288150			
		69203.1			68173.2			
		81805.3			133820			
		32269.2			148583			
		155756			239609			
		123270			276197			
		23318.6			630580			
		115767			539988			

Tabla A.20: Resultados de la ejecución de GACEPv6

GACEPv6								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	202996	**200**	25	398105	**300**	25	388648
		87154.7			955259			30576.3
		200799			319902			280124
		233498			30919.5			456889
		237732			103771			15671.4
		79612.2			811711			267206
		10605.3			42530			495371
		67661.9			730334			9559.62
		239530			813434			249094
		27234.7		50	658044		50	1019890
	50	20351.8			51063.5			510577
		57588.9			211043			109755
		3857.9			244109			896343
		52419.5			227337			319409
		61651.4			193643			736668
		39338.1			84287.4			752835
		15486.3			61736.7			46399.4
		21755.7			153220			785110
		37758			52188.5			774815
		94130.8		75	296119			
	75	88522.7			383598			
		110194			220043			
		37301.4			235922			
		108710			236252			
		61564.1			487782			
		17464.7			437910			
		56696.8			230042			
		59292.9			330402			
		74719			110881			
		150678		100	150263			
	100	191492			460300			
		105001			295679			
		66503.2			65862.8			
		80308			136923			
		29851.6			148548			
		151456			243658			
		119540			282428			
		20972.6			622414			
		112587			539184			

Tabla A.21: Resultados de la ejecución de GACEPv7

GACEPv7								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	202996	**200**	25	398105	**300**	25	388648
		87154.7			955259			30576.3
		200799			319902			280124
		233498			30919.5			456889
		237732			103771			15671.4
		79612.2			811711			267206
		10605.3			42530			495371
		67661.9			730334			9559.62
		239530			813434			249094
		27234.7		50	658044		50	1019890
	50	20351.8			51063.5			510577
		57588.9			211043			109755
		3857.9			244109			896343
		52419.5			227337			319409
		61651.4			193643			736668
		39338.1			84287.4			752835
		15486.3			61736.7			46399.4
		21755.7			153220			785110
		37758			52188.5			774815
		94130.8		75	296119			
	75	88522.7			383598			
		110194			220043			
		37301.4			235922			
		108710			236252			
		61564.1			487782			
		17464.7			437910			
		56696.8			230042			
		59292.9			330402			
		74719			110881			
		150678		100	150263			
	100	191492			460300			
		105001			295679			
		66503.2			65862.8			
		80308			136923			
		29851.6			148548			
		151456			243658			
		119540			282428			
		20972.6			622414			
		112587			539184			

Tabla A.22: Resultados de la ejecución de GACEPv8c50

GACEPv8c50								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	208526	**200**	25	390278	**300**	25	383330
		90690.4			966574			31839.5
		206512			318405			272530
		240222			34156.3			456556
		243711			109968			16485
		82864.9			824503			256325
		11912.7			45618.3			497786
		70754			731825			10045.7
		243397			822726			240092
	50	27531.4		50	665845		50	1028700
		20893.1			51320.2			489957
		59031.5			213012			111653
		4147.06			246547			897896
		53874.6			229966			309268
		62534.1			195339			735155
		39075.3			82257.1			748129
		16335			61879.7			47902
		22544.1			152429			780738
		38241			52286.8			775321
	75	96221.3		75	292351			
		90184.9			388743			
		112028			213048			
		38660.4			230454			
		111249			232060			
		61184.8			493743			
		18604.4			444441			
		57487.4			224692			
		60342.1			329126			
		75682.9			112423			
	100	154667		100	153628			
		191955			458351			
		106202			288870			
		68969.5			67663.3			
		81930			137380			
		32302.6			148861			
		155830			241430			
		123520			275664			
		23540.2			627975			
		115776			542740			

Tabla A.23: Resultados de la ejecución de GACEPv8c60

GACEPv8c60								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	206772	**200**	25	388376	**300**	25	381354
		90510.3			966376			31334.7
		206646			316396			272151
		238715			33376.8			454416
		242901			110308			16588.4
		81999.7			823646			257092
		11389			44555.3			497473
		69783.6			730355			10033.5
		242551			821771			239119
		27291.4		50	661643		50	1027210
	50	21092.3			51549.3			485166
		58995.8			212099			111682
		4069.48			246164			897908
		53746.7			229445			309869
		62519.6			194198			733747
		39136.8			83216			743000
		16210.8			61889.9			47859.9
		22471.4			152019			776317
		38228.5			52288.6			775052
		95032.3		75	291996			
	75	90529.2			387162			
		112395			214501			
		38042			228536			
		111097			227793			
		62009.5			493315			
		18821.2			443712			
		58104.8			224596			
		59788.1			328795			
		74665.1			111592			
	100	153672		100	152346			
		191604			458321			
		105690			287103			
		68544.9			68209.4			
		80793.2			134196			
		31799.8			149518			
		155364			240437			
		122633			279535			
		22913.1			628102			
		116654			540962			

Tabla A.24: Resultados de la ejecución de GACEPv8c70

GACEPv8c70								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	206713	**200**	25	390558	**300**	25	379916
		88814.8			962185			31087.2
		206306			315567			266377
		238898			32652.5			451294
		241434			107607			16226.6
		81624.5			818438			257624
		11451.8			43534			495414
		68523.5			728491			9974.94
		242618			817310			236016
		26685.4		50	653443		50	1024970
	50	20751.6			50813.4			489928
		58679.7			212041			111513
		4013.84			245839			892044
		53712.3			229126			308127
		62550.2			194396			723167
		39220.4			82334.8			735297
		15988.8			61418			47395.5
		22280.8			149779			770496
		37657.4			51676.8			769369
		95140.6		75	286527			
	75	89570.9			385079			
		112135			214957			
		37705.1			226341			
		109959			225192			
		60622			492004			
		18187.6			443041			
		56405.6			222210			
		59185.5			326301			
		74057.3			112244			
	100	152488		100	152072			
		191553			455230			
		104446			285382			
		67854.6			67137			
		80108			135438			
		31151.6			148486			
		154773			240327			
		120755			275166			
		22179.2			624253			
		113935			537410			

Tabla A.25: Resultados de la ejecución de GACEPv8c80

GACEPv8c80								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	205627	**200**	25	386180	**300**	25	377838
		88089.2			960457			30664.6
		201824			316395			267740
		234574			31608.3			448588
		238990			105604			15918.5
		80341.4			813256			256219
		11001.6			43174.5			492340
		67193.1			718573			9692.4
		240068			807907			240307
		26308.3		50	641372		50	1014740
	50	20057.8			50157.9			489088
		58155.7			209043			110140
		3922.22			244132			886688
		52805.6			227542			308080
		62204.2			192147			720509
		38292.4			81519.7			733485
		15795.4			60632.1			46861
		21935.5			147891			762039
		36837.2			51303.2			760492
		94751.2		75	285241			
	75	87614.4			381938			
		110816			208501			
		37110.4			225470			
		109869			224438			
		59500.5			488482			
		17694			439801			
		55403.2			220481			
		57946.8			322838			
		73588.3			110627			
		151751		100	150290			
	100	191543			446568			
		101779			281318			
		66257			65991.6			
		78126.4			133963			
		30212.7			146249			
		152633			236802			
		119635			274319			
		21687.7			617135			
		112290			528967			

Tabla A.26: Resultados de la ejecución de GACEPv9c50

GACEPv9c50								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	203358	**200**	25	393503	**300**	25	384866
		86930.3			955148			30571.5
		199771			317344			276555
		232834			30676.4			453347
		236147			105226			15566.6
		80960.8			813202			263853
		10560.9			42665.9			494573
		66861.8			721422			9548.38
		238911			810671			247922
		27116.7		50	648885		50	1014950
	50	19877.7			50178.7			499872
		57605.8			210881			109833
		3802.82			243188			891463
		52360.4			227298			318661
		61729.2			193264			729668
		38489.1			84132.3			742593
		15418.3			61578.5			45718.2
		21499.5			151667			774435
		37381.6			51316.7			769503
		93218.3			292338			
	75	89147.6		75	381945			
		108703			217925			
		36268.5			234074			
		108298			232054			
		60815.8			487496			
		17268.5			437852			
		56667.7			226845			
		58147.7			326266			
		73666.6			110995			
	100	149623		100	149326			
		191090			454945			
		104282			294687			
		66047.6			66036.4			
		79251			136770			
		29408.1			146409			
		150436			241560			
		119946			281410			
		21070.5			618351			
		112890			535204			

Tabla A.27: Resultados de la ejecución de GACEPv9c60

GACEPv9c60								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	200048	**200**	25	392594	**300**	25	381635
		86753.3			955156			30440.2
		197967			314735			276108
		230749			31113			452201
		235636			103184			15511.1
		79229.8			809623			263687
		10579.2			42648			493392
		66613.2			723401			9562.14
		239451			806866			249108
		26453.7		50	646935		50	1015280
	50	19854.1			50407.4			508106
		57480.5			209467			109296
		3834.68			242630			893534
		52140.2			226325			317478
		61681			192637			725674
		38703.1			83375.9			738187
		15318.9			61215.3			45785.7
		21307.7			151732			775720
		37491.1			51282.8			766823
		93087.2		75	291854			
	75	88785.4			379793			
		109110			218514			
		35865.1			231903			
		108270			234237			
		60762.4			486394			
		17001.9			436859			
		56477.1			224934			
		57757.9			324286			
		72953.9			109703			
		149058		100	147687			
	100	191502			455862			
		102278			296514			
		65097.1			65742.1			
		78636.2			134742			
		29356			145705			
		150366			241045			
		118168			280671			
		20407.8			616596			
		112348			531363			

Tabla A.28: Resultados de la ejecución de GACEPv9c70

GACEPv9c70								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	201570	**200**	25	390354	**300**	25	378906
		85629.7			952449			29753.9
		197783			312448			275416
		232730			30001.7			449956
		235682			103813			15441.5
		78551.8			805306			266418
		10513.4			42447.6			490569
		65257.6			715737			9528.66
		238359			802817			247514
	50	26247		50	642994		50	1012040
		19540.7			49702.9			505505
		57479.4			208380			108104
		3766.74			242256			887410
		51778.7			225644			313868
		61648			190572			721449
		37766.6			82472.1			730217
		15083.9			60858.5			45139
		21115.2			150575			770346
		36954.4			51231.1			762922
	75	91910.2		75	287878			
		87177			379786			
		108439			216835			
		35374.4			230282			
		107847			231299			
		59636.8			486178			
		17232.1			435635			
		55790.8			225570			
		57375			324817			
		73033.1			108746			
	100	148525		100	147490			
		191110			451466			
		102468			292975			
		65358			65020.8			
		77969.1			133484			
		29200.8			145032			
		150145			239219			
		117624			278266			
		20552.4			612741			
		111393			531882			

Tabla A.29: Resultados de la ejecución de GACEPv9c80

GACEPv9c80								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	199223	**200**	25	387021	**300**	25	374665
		84413.4			949957			29822.3
		195766			311427			272678
		230580			30284.6			445988
		233189			103140			15184.2
		77109			799222			266890
		10504.9			41942.8			489369
		64134.5			709829			9544.12
		237987			795168			250876
		25676.1		50	635883		50	1005680
	50	19340.5			49405.7			505389
		57508.2			207698			107522
		3773.88			241737			881873
		51433.9			224784			310766
		61670			189618			716271
		37370.5			81605.5			724768
		15115.6			60079.8			44918.1
		21065.7			148676			761738
		36329.1			50320.9			753019
		91549.9		75	285356			
	75	86631.1			375998			
		107614			215262			
		35563.8			229616			
		107835			229455			
		58266.1			485378			
		16679.3			434048			
		54996.9			222548			
		56622.5			317953			
		71704			107675			
		147774		100	146317			
	100	190947			443220			
		99399.7			288824			
		64263			63555			
		75780.7			132066			
		29087.9			142950			
		148359			235358			
		116339			275669			
		20393.5			609990			
		108752			522567			

Tabla A.30: Resultados de la ejecución de GACEPv10

GACEPv10								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	206380	**200**	25	405301	**300**	25	394303
		90338.9			963788			31820.3
		204372			325040			287827
		238467			32741.7			459853
		242406			106731			16303.2
		83994.8			825236			271799
		11542.8			43194.3			498003
		69336.6			741029			9824.9
		242925			823044			255253
		28096.3		50	669751		50	1023580
	50	21059.1			51665.5			515281
		58190.3			212668			112598
		4165.42			246237			903563
		53280.8			227989			320454
		61988			195911			751439
		39703.6			86683.7			760057
		16219.9			63456.8			48340
		22899.8			157225			794399
		38589.9			53144.3			788242
		95455.6		75	299837			
	75	90582			387344			
		112348			226450			
		38978.4			240024			
		110714			241247			
		63073			491881			
		18355.6			442820			
		59165.7			235575			
		60848.3			336276			
		76050.5			114125			
		153164		100	154017			
	100	191633			468504			
		107684			300655			
		69132.1			67173.7			
		83364.8			139601			
		31740.9			149263			
		154694			245604			
		122587			285941			
		22510.7			625929			
		116847			545551			

Tabla A.31: Resultados de la ejecución de GACEPv11c50

GACEPv11c50								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	205638	**200**	25	398388	**300**	25	384918
		91250.6			963022			31494.5
		204911			322718			278693
		238185			32475.9			458597
		242417			106874			16192.8
		83001.6			822255			266561
		11527			43929.1			497730
		69456.8			731403			9888.36
		240823			819634			246268
		27878.1		50	660976		50	1022150
	50	21340.9			51498			501406
		58241.9			212267			112156
		4046.5			246048			902487
		53512.8			228121			321510
		62064.4			195194			739994
		39574.1			84713			751352
		15960.9			62649.9			47547.1
		22283.2			154634			786058
		38442.4			52414.2			780290
		95467.3		75	296077			
	75	90356.1			387546			
		111887			222354			
		38041.9			234269			
		109931			235757			
		61855			490961			
		18314.5			442128			
		58360.2			228918			
		59858.8			331122			
		75450			112549			
		152902		100	153194			
	100	191843			462973			
		106822			298925			
		68916.8			67194.8			
		81932.9			138481			
		31700.8			148856			
		153961			244831			
		122734			283510			
		22244.9			628434			
		115182			538755			

Tabla A.32: Resultados de la ejecución de GACEPv11c60

GACEPv11c60								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	204849	**200**	25	398223	**300**	25	388200
		88546.5			958550			31156.4
		202638			322822			278808
		234615			32252.3			458071
		240364			105668			16051.2
		82501.1			816754			265934
		11607.5			43647.7			496704
		68637.1			731369			9874.06
		240966			814223			246617
		27454.7		50	659673		50	1022960
	50	20581			51009.7			503403
		58060.9			211571			111941
		3961.52			245388			900348
		52724.8			228003			319870
		62164.8			195628			741364
		39357.1			84753.8			745092
		15908.4			62819.5			46883.3
		22260.4			153867			786025
		37965.7			52445.3			781417
		95334.3		75	295943			
	75	90284			387216			
		110580			220738			
		37651.8			232970			
		109934			235239			
		62106.4			490691			
		18112.2			441462			
		58096.6			230193			
		59717.5			332652			
		75094.2			113075			
		151688		100	152379			
	100	191712			465460			
		106694			299458			
		68875.5			66613.2			
		81858.2			138309			
		31321			149232			
		153221			245579			
		121482			286119			
		21596.1			623141			
		114925			540669			

Tabla A.33: Resultados de la ejecución de GACEPv11c70

GACEPv11c70								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	203811	**200**	25	396441	**300**	25	386012
		88498.2			957029			30726.3
		201480			318305			277792
		233837			31519.7			454882
		237803			105965			15951.5
		80200.1			815161			264403
		11309.3			42811.8			495899
		68369.2			729939			9696.2
		240340			817563			248291
		27184.1		50	657773		50	1019310
	50	20288.2			50759.5			509867
		57725.1			210991			109951
		3916.32			244736			897648
		52664.3			227639			318844
		61906.4			194226			735509
		38943.3			84481.3			748928
		15678.9			62351.8			46234.9
		22062.4			153196			781894
		37764.9			51722.5			777148
		94024.7		75	296350			
	75	88778.9			384748			
		110536			220155			
		37170.2			234177			
		108811			235628			
		62042.7			488627			
		17457.1			440451			
		57220.7			229520			
		58767.8			329468			
		74199.5			111884			
		151213		100	151155			
	100	191703			458497			
		104153			296323			
		67623.8			65888.9			
		79819.7			136224			
		30203.6			147859			
		152331			242843			
		120938			282462			
		21678.3			621909			
		112717			537738			

Tabla A.34: Resultados de la ejecución de GACEPv11c80

GACEPv11c80								
n	densidad	Resultado	n	densidad	Resultado	n	densidad	Resultado
100	25	202779	**200**	25	393393	**300**	25	382847
		86559.8			955731			30447.6
		199729			315249			277777
		231206			30907			451229
		236331			104747			15654.6
		79915.8			806752			267486
		10904.1			42714.3			492804
		66139.5			724470			9604.22
		238841			806500			249939
		26332.4		50	648645		50	1011950
	50	19926.1			50103			510320
		57559.7			210000			109518
		3869.26			243071			890660
		52024.9			226706			314790
		61684.4			192171			731202
		38185			83263.7			740387
		15214.2			60999			45607
		21361.1			151605			772325
		37312.5			51222			771114
		93398.2		75	291861			
	75	88137.5			380357			
		109072			217844			
		36227.7			232259			
		108248			233964			
		59927.6			486873			
		17248.6			437733			
		55611.5			225800			
		57811.3			325266			
		73639.4			109902			
		150100		100	148619			
	100	191540			454785			
		103653			295082			
		64939.6			65356.1			
		78179.7			135994			
		29699.9			144303			
		150956			239622			
		120382			276812			
		20787.6			615525			
		111613			531534			