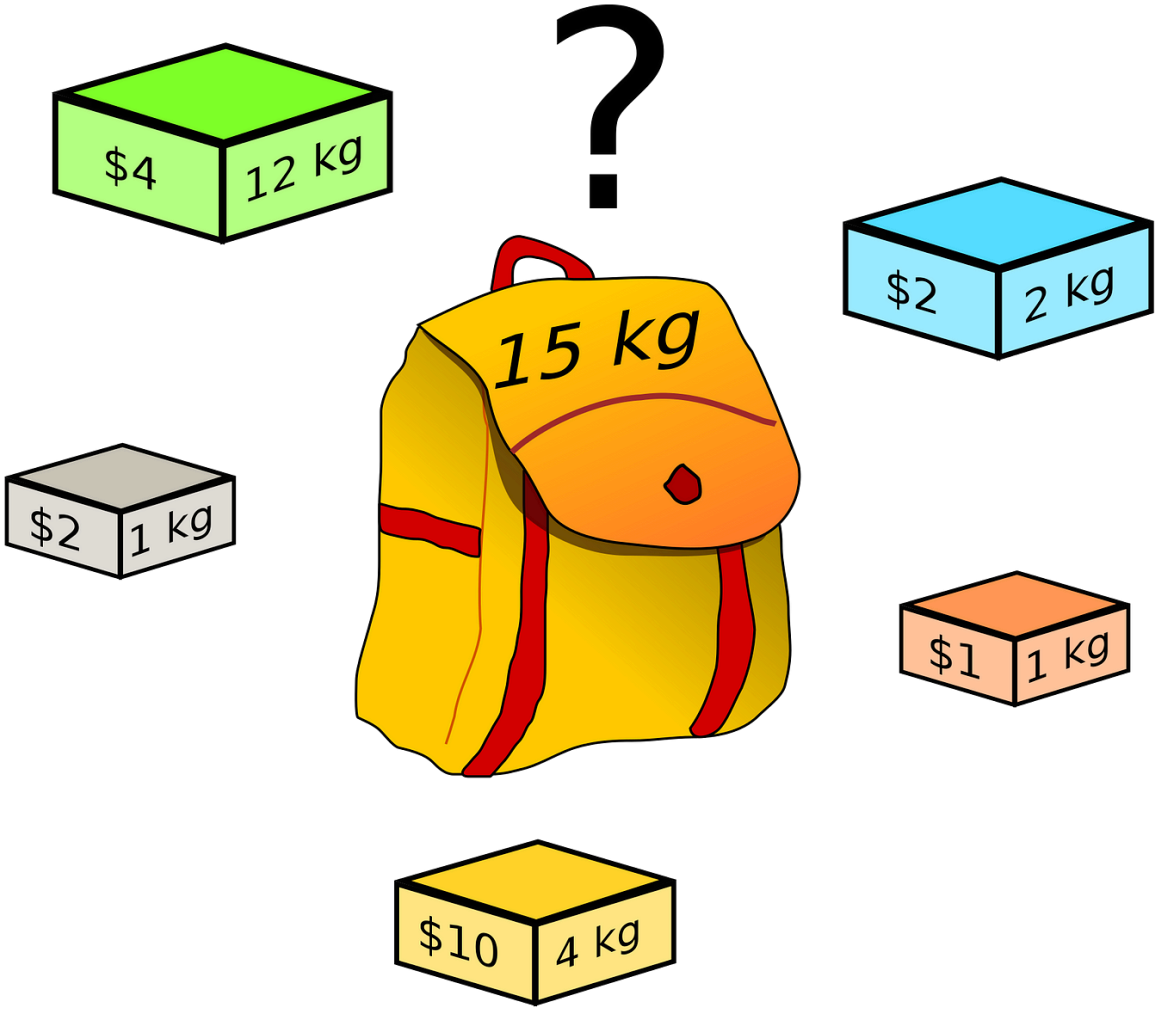
Práctica 2
Resolución del Problema de la Mochila Cuadrática mediante algoritmos Genéticos y Meméticos



Santiago Herron Mulet

Metaheurística Grupo de Prácticas 3

ÍNDICE

1. Descripción del problema.

a. Planteamiento:

El problema de la mochila cuadrática (QKP) es un problema de optimización combinatoria NP-completo que busca maximizar el beneficio obtenido al seleccionar un conjunto de objetos para colocar en una mochila con capacidad limitada. Cada objeto tiene un peso, un valor y un beneficio adicional que se obtiene si se seleccionan dos objetos específicos. El objetivo es encontrar la combinación de objetos que maximice el beneficio total sin exceder la capacidad de la mochila.

b. Formulación matemática:

i. Función a maximizar f(x):

$$\text{maximize} \left\{ \sum_{i=1}^{n} p_i x_i + \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} P_{ij} x_i x_j : x \in X, x \text{ binary} \right\}$$

ii. Sujeta a:

$$\text{subject to } X \equiv \left\{ x \in \{0,1\}^n : \sum_{i=1}^{n} w_i x_i \leq W; x_i \in \{0,1\} \text{ for } i = 1, \ldots, n \right\}$$

donde:
p_i: Valor del objeto i.
p_ij: Beneficio adicional obtenido al seleccionar los objetos i y j juntos.
w_i: Peso del objeto i.
W: Capacidad de la mochila.
x_i: Variable binaria que indica si el objeto i se selecciona (x_i = 1) o no (x_i = 0).

c. Propiedades y complejidad:

El QKP es un problema NP-completo, lo que significa que no existe un algoritmo conocido que pueda resolverlo de manera eficiente para todas las instancias. A medida que aumenta el tamaño del problema, la complejidad computacional para encontrar la solución óptima aumenta exponencialmente.

d. Algoritmos de resolución:

Existen diversos algoritmos para resolver el QKP, los cuales se pueden dividir en dos categorías principales:

i. Algoritmos exactos:

Estos algoritmos garantizan encontrar la solución óptima, pero pueden ser muy lentos para problemas de gran tamaño.

ii. Algoritmos heurísticos:

Estos algoritmos no garantizan encontrar la solución óptima, pero son mucho más rápidos que los algoritmos exactos.

2. Descripción de los ámbitos generales de los AGs y AMs.

a. A nivel teórico.

i. Naturaleza heurística:

1. AG y AM son algoritmos heurísticos que ofrecen soluciones aproximadas para problemas complejos como la mochila cuadrática, sin garantizar la solución óptima.
2. Estos algoritmos son eficaces en la búsqueda y optimización de soluciones en espacios de búsqueda amplios y complejos.

ii. Exploración y explotación:

1. Ambos algoritmos combinan exploración, mediante la generación aleatoria y combinación de soluciones, con explotación, refinando soluciones prometedoras mediante operadores genéticos como el cruce y la mutación.

iii. Naturaleza estocástica y exploratoria:

1. Utilizan aleatoriedad en la selección, cruce y mutación de soluciones para explorar el espacio de búsqueda y evitar quedar atrapados en óptimos locales.
2. Esta naturaleza estocástica les permite encontrar soluciones prometedoras en problemas complejos al permitir una exploración más amplia del espacio de búsqueda.

iv. Iteraciones y convergencia:

1. Ambos algoritmos evolucionan a través de múltiples generaciones, donde cada generación implica la aplicación de operadores genéticos para generar nuevas soluciones.

2. La convergencia se alcanza cuando la población converge hacia una solución que satisface las restricciones del problema y maximiza la función objetivo, aunque no necesariamente la solución óptima global.

v. Parámetros y ajustes:

1. Los AG y AM tienen parámetros que afectan su comportamiento y rendimiento, como el tamaño de la población, la tasa de cruce y la tasa de mutación.
2. La configuración adecuada de estos parámetros es crucial para equilibrar la exploración y la explotación del espacio de búsqueda y mejorar el rendimiento del algoritmo en el problema de la mochila cuadrática.

b. Esquema de representación de soluciones:

En el problema de la mochila cuadrática, cada individuo de la población se representará como un objeto de la clase Mochila. Esta clase contendrá un atributo asignacion, que será un vector booleano. Cada elemento del vector representará si el objeto correspondiente está presente (true) o ausente (false) en la mochila. Por ejemplo, si el índice i es true, significa que el objeto i ha sido seleccionado para ser incluido en la mochila; de lo contrario, no ha sido seleccionado.

Además, cada mochila también tendrá los atributos peso y beneficio. El peso representará la suma de los pesos individuales de los objetos incluidos en la mochila, mientras que el beneficio será el beneficio total asociado a esa mochila, el cual se detallará en la siguiente sección.

Este esquema de representación facilitará la definición de la población, que estará compuesta por un vector de objetos de la clase Mochila. Cada elemento del vector representará a un individuo de la población.

c. Descripción de la función objetivo (fitness):

En nuestro problema de la mochila, el fitness vendrá representado por el beneficio de la mochila, es decir, la suma de los beneficios individuales de cada objeto más los beneficios por interdependencia de cada unos de los objetos incluidos entre sí. En mi caso, he incluído dicho cálculo del fitness en un método de la clase mochila. Como se observa en el siguiente pseudocódigo, he añadido también a esta función el cálculo del peso de la mochila para no añadir complejidad añadida a su cálculo:

```
Función Fitness():
    beneficio = 0
```

```
  peso = 0
  Para cada índice i en el rango de 0 a tamaño de asignacion - 1:
    Si asignacion[i] es verdadero:
      beneficio += b_individual[i]
      peso += pesos[i]
      Para cada índice j en el rango de 0 a tamaño de asignacion - 1:
        beneficio += b_interdependencias[i][j] * asignacion[j]
  Devolver beneficio
```

d. Descripción de operadores comunes:

i. Operador de generador de vecinos para BL en algoritmos meméticos.

Se usará el mismo generador de vecinos que en la anterior práctica en el que se consideran soluciones vecinas aquellas soluciones factibles en las que sólo varíe desde la solución inicial una permutación de un objeto incluído por otro que no lo estaba:

```
Función VecinosPermutacion(actual, W):
  vecinos = nuevo vector
  v = obtener asignación de actual

  peso_actual = obtener peso de actual
  Para cada índice i en el rango de 0 a tamaño de v - 1:
    Si v[i] es verdadero:
      Para cada índice j en el rango de 0 a tamaño de v - 1:
        Si no v[j] y i distinto de j:
          Si peso_actual - peso del objeto i + peso del objeto j <= W:
            Agregar  (i, j) a vecinos
  Devolver vecinos
```

ii. Creación de población inicial.

La población inicial se creará de manera totalmente aleatoria, es decir, se crearán tantos individuos como el tamaño de la población y cada individuo tendrá unos genes aleatorios:

```
Función solucionInicial(n, capacidad):
  s = nuevo vector de tamaño n, inicializado a falso
  o = una copia del vector de objetos

  Mientras o no esté vacío:
    Mezclar aleatoriamente los objetos en o
```

```
        Si capacidad >= peso del último objeto en o y el último objeto no está en
s:
            Reducir capacidad en el peso del último objeto en o
            Establecer el último elemento en s a verdadero
        Eliminar el último objeto en o

    Devolver s

Función poblacionInicial(n, capacidad, tam_poblacion):
    pobInicial = nuevo vector de Mochilas
    Para cada índice i en el rango de 0 a tam_poblacion - 1:
        Agregar solucionInicial a pobInicial
    Devolver pobInicial
```

iii. Mecanismo de selección en los AGs.

Para seleccionar entre individuos en los algoritmos genéticos (la aplicación de está selección se explicará en la sección correspondiente a los algoritmos genéticos) se hará mediante torneo entre tres individuos.

```
Función torneo(m1, m2, m3):
        Devolver m con mejor Fitness
```

iv. Operador de cruce dado (dos puntos).

Para el cruce de dos individuos con el objetivo de la generación de dos sucesores se nos proporcionó el cruce dos puntos. Este cruce consiste en seleccionar de manera aleatoria un segmento de los genes. A continuación se creará el hijo 1 igual que el padre 1 menos el segmento que será del padre 2 y el hijo 2 viceversa:

```
Función cruceDosPuntos(s1, s2, W):
    hijo1 = copiar s1
    hijo2 = copiar s2
    p1 = entero aleatorio en el rango de 0 a tamaño de s1 - 1
    p2 = entero aleatorio en el rango de p1 a tamaño de s1 - 1


    Para cada índice i en el rango de 0 a tamaño de s1 - 1:
        Si i está en el rango [p1, p2):
            hijo1[i] = s2[i]
            hijo2[i] = s1[i]

            Ajustar factibilidad hijos de manera aleatoria
Devolver hijos
```

v. Operador de mutación:

Para obtener diversidad con el objetivo de aumentar el espacio de búsqueda de soluciones se aplican mutaciones a los individuos con una cierta probabilidad. Esta mutación consiste simplemente en alterar un gen.

```
función mutacion(c, peso_actual, W):

   mientras no factible:
      gen_mutar := aleatorio(0, tamaño_de_c - 1)
      si no s_mut[gen_mutar] y (peso_actual + pesos[gen_mutar]) > W:  // 
Verificar si la mutación es factible
         factible := falso
      sino:
         factible := verdadero

   retornar gen_mutar  // Devolver el índice del gen a mutar
```

e. Otras implementaciones comunes.

i. Lectura desde ficheros:

Se ha implementado una función de lectura para obtener los datos del problema desde un archivo externo. El programa espera dos argumentos como mínimo: el algoritmo a utilizar y la ruta del archivo. Luego, lee el número de objetos, la capacidad de la mochila, los pesos, los beneficios individuales y los beneficios por interdependencias desde el archivo. Estos datos se utilizan para crear un vector de objetos que representan los elementos del problema, que luego se utilizan en el algoritmo seleccionado para resolver el problema de la mochila cuadrática.

ii. Cálculo de tiempos de ejecución:

Para calcular los tiempos de ejecución de los algoritmos, se utiliza la biblioteca <chrono>. Después de cargar los datos del problema desde el archivo, se ejecuta el algoritmo correspondiente. Se toma el tiempo de inicio antes de la ejecución del algoritmo y el tiempo de finalización después de que el algoritmo haya terminado su ejecución. Luego, se calcula la diferencia entre estos dos tiempos para obtener el tiempo de ejecución total del algoritmo.

3. Algoritmos Genéticos

a. Estructura general.

i. Comprobación de la condición de parada:

Comprobar que no se han superado las 90000 evaluaciones del fitness.

ii. Selección:

Se seleccionan qué individuos de la población se van a considerar para la evolución de la población.

iii. Cruce:

Con una probabilidad, se cruzan los individuos seleccionados.

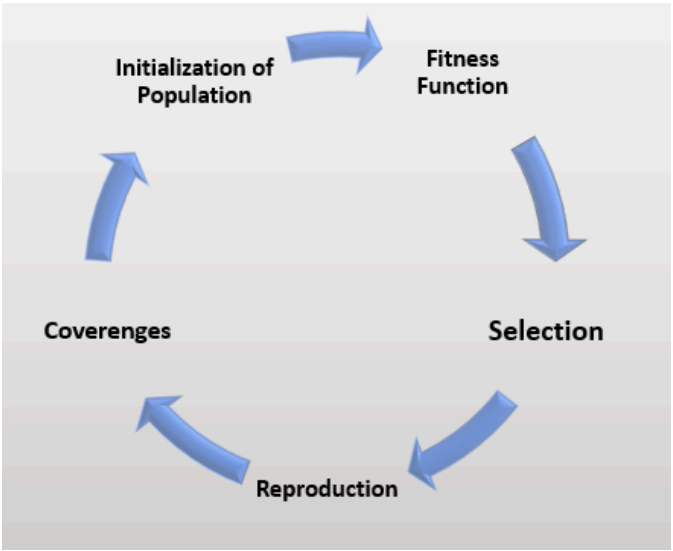
iv. Mutación.

Con una probabilidad, se mutan los individuos cruzados.

v. Reemplazo:

Se inicializa la siguiente población con las modificaciones de esta iteración.

vi. Volver a 1.



b. Propuesta de cruce.

A parte de la propuesta de cruce ya explicada anteriormente (cruce dos puntos), se desarrolló una nueva propuesta de cruce que utiliza conocimiento del problema. En concreto el cruce funciona de la siguiente manera. Primero, para el cruce en sí, se usa el cruce uniforme, es decir, para cada gen de los padres se tira un dado para elegir si el gen del padre 1 va al hijo 1 o al hijo 2, y el gen del padre 2 irá al que no fue seleccionado. A continuación, aquí es donde se aplica el conocimiento, para

ajustar la factibilidad de los hijos, se irán quitando los objetos en orden decreciente de peso (se eliminará primero el objeto más pesado). Esto es con el objetivo de crear soluciones factibles con mayor diversidad.

```
Función cruceUniformePesado(c1, c2, W):
    s1 = Asignación de c1
    s2 = Asignación de c2
    hijo1 = Nuevo vector de booleanos del tamaño de s1
    hijo2 = Nuevo vector de booleanos del tamaño de s2

    Para cada índice i en el rango de 0 a tamaño de s1 - 1:
        Si aleatorio entre 0 y 1 es 0:
            hijo1[i] = s1[i]
            hijo2[i] = s2[i]
        Sino:
            hijo1[i] = s2[i]
            hijo2[i] = s1[i]

    Mientras el peso de hijo1 sea mayor que W:
        Eliminar el objeto más pesado de hijo1

    Mientras el peso de hijo2 sea mayor que W:
        Eliminar el objeto más pesado de hijo2
```

Ejemplo de uso:
padre1 = 010100
padre2 = 000111

hijo1 = 000110 -> índices padre1 = 1, 4, 6, índices padre2 = 2, 3, 5
hijo2 = 010101 -> índices padre1 = 2, 3, 5, índices padre2 = 1, 4, 6

hijo1 no se pasa de peso entonces se queda igual.
hijo2 se pasa se peso, se quita su objeto más pesado -> índice 2
hijo2 = 000101

c. Algoritmo genético generacional.

Las particularidades del generacional respecto a la estructura general estarán relacionadas con que en este algoritmo se trabaja en cada iteración con toda la población por lo que:

i. Se seleccionarán mediante torneo entre 3 individuos aleatorios de la población inicial tantas veces como tamaño de la población.
ii. De la población seleccionada, se cruzarán individuos dependiendo de la probabilidad de cruce.

iii. De la población cruzada, se mutarán individuos dependiendo de la probabilidad de mutación.
iv. Para el reemplazo, se considerará la nueva población la población mutada reemplazando el mejor individuo de la población anterior en el caso de ser mejor que el mejor individuo de la nueva.

Considerando este algoritmo, tenemos que en cada iteración se calculará tantas veces la función objetivo como tamaño de la población.

Consideraciones sobre aleatoriedad para reducir complejidad:
Este algoritmo depende mucho de la aleatoriedad ya que esta es necesaria para la selección (elección de individuos aleatorios), el cruce (probabilidad) y la mutación (probabilidad). Podríamos generar números aleatorios cada vez que los precisáramos pero de esta manera aumentaría en gran medida la complejidad del algoritmo. Por este motivo, sólo se generan números aleatorios en la selección. Sin embargo la aleatoriedad se mantiene ya que calcularemos tanto los cruces, como las mutaciones esperadas de la población teniendo en cuenta la probabilidad para estimar el número de cruces y mutaciones esperados y como la población seleccionada está ordenada aleatoriamente la aleatoriedad no se verá afectada.

```
Función AGG:
   iteracion = tam_poblacion

   Mientras iteracion sea menor o igual a max_iteraciones:
      p_nueva = poblacion_actual

      Para cada índice i en el rango de 0 a tam_poblacion - 1:
         p_nueva[i] = torneo(poblacion_actual[aleatorio], poblacion_actual[aleatorio],
poblacion_actual[aleatorio])

      cruces_esperados = (tam_poblacion / 2) * prob_cruce
      Para cada índice i en el rango de 0 a cruces_esperados - 1:
         cruce(p_nueva[i], p_nueva[i + 1], W)

      mutaciones_esperadas = tam_poblacion * prob_mutacion
      Para cada índice i en el rango de 0 a mutaciones_esperadas - 1:
            gen_mutar = Aleatorio entre 0 y genes - 1
            p_nueva[i].mutacion(gen_mutar, W)

      Para cada mochila en p_nueva:
         Calcular su fitness

      mejorPadre = La mochila con el mayor fitness en poblacion_actual
      peorHijo = La mochila con el menor fitness en p_nueva
      mejorHijo = La mochila con el mayor fitness en p_nueva

      Si Fitness de mejorPadre es mayor que Fitness mejorHijo:
         Reemplazar peorHijo en p_nueva con mejorPadre
```

```
        Copiar p_nueva en poblacion_actual

        iteracion += tam_poblacion

    Devolver la asignación de la mochila con el mayor fitness en poblacion_actual
```

d. Algoritmo genético estacionario.

Las particularidades del algoritmo estacionario estarán relacionadas con que en este caso, en cada iteración del algoritmo, sólo se trabajará sobre 2 individuos.
Provocando que:
i. En la selección se harán 2 torneos entre 3 individuos elegidos aleatoriamente en cada torneo.
ii. Para el cruce, se cruzarán siempre (aquí no hay probabilidad) los 2 ganadores del torneo únicamente resultando en 2 nuevos individuos.
iii. Para la mutación, se mutarán estos 2 individuos con una probabilidad de mutación.
iv. Para el reemplazo, los dos peores individuos de la población actual competirán con los dos nuevos individuos por integrarse en la población, es decir, de esos 4, permanecerán los 2 mejores.

```
Función AGE:
    iteracion = tam_poblacion
    Mientras iteracion sea menor que max_iteraciones:
        h1 = torneo(poblacion_actual[aleatorio], poblacion_actual[aleatorio],
poblacion_actual[aleatorio])
        h2 = torneo(poblacion_actual[aleatorio], poblacion_actual[aleatorio],
poblacion_actual[aleatorio])
        cruce(h1, h2, W)
        Si Aleatorio entre 0.0 y 1.0 es menor que prob_mutacion:
            gen_mutar = Aleatorio entre 0 y genes - 1
            h1.mutacion(gen_mutar, W)
        Si Aleatorio entre 0.0 y 1.0 es menor que prob_mutacion:
            gen_mutar = Aleatorio entre 0 y genes - 1
            h2.mutacion(gen_mutar, W)
        Calcular fitness de h1 y h2
        Ordenar poblacion_actual por fitness de menor a mayor
        Cambiar los dos últimos individuos de poblacion_actual con los dos mejores de
{h1, h2, peor de poblacion_actual, segundo peor de poblacion_actual}
        Incrementar iteracion en 2
    Devolver la asignación de la mochila con el menor fitness en poblacion_actual
```

4. Algoritmos meméticos.

a. Estructura general.

i. Comprobación de la condición de parada (90000 evaluaciones).
ii. Aplicación de un algoritmo genético un número determinado de evaluaciones o generaciones. (Siempre que no se supere el máximo de evaluaciones).
iii. Aplicación de algoritmo de búsqueda local sobre un determinado conjunto de individuos. (Siempre que no se supere el máximo de evaluaciones).
iv. Volver a 1.

b. Algoritmo:

```
Función AM:
    iteracion = tam_poblacion
    Mientras iteracion sea menor que max_iteraciones:
        max = aplicacion_BL * tam_poblacion si iteracion + aplicacion_BL *
tam_poblacion es menor o igual que max_iteraciones, de lo contrario max es
max_iteraciones - iteracion
        it = 0
        genetico(poblacion_actual, genes, W, it, max, prob_cruce, prob_mutacion,
tam_poblacion, cruce)
        iteracion += max
        bl_esperado = tam_poblacion * prob_aplicacion_BL
        Si BL_elitista es verdadero, ordenar poblacion_actual por fitness de menor a
mayor
        Para cada índice i en el rango de 0 a bl_esperado - 1 y mientras iteracion sea
menor que max_iteraciones:
            iteraciones_BL = 0
            Aplicar búsqueda local a poblacion_actual[i] con genes, W, max_iteraciones,
iteraciones_BL, vecinos, genes
            Incrementar iteracion por iteraciones_BL
    Devolver la asignación de la mochila con el mayor fitness en poblacion_actual
```

c. Variantes algoritmo memético.

Gracias a la implementación desarrollada, las ejecuciones de las diferentes variantes del algoritmo memético se ejecutan en la misma función cambiando solo tres parámetros:

i. aplicacion_BL: entero que define cada cuantas generaciones del algoritmo genético generacional se va a ejecutar el BL.

ii. prob_aplicacion_BL: double que define el porcentaje del tamaño de la población al que se le va a aplicar el BL.

iii. BL_elitista: booleano que determina si el bl se aplica sobre los mejores individuos o sobre individuos elegidos de manera aleatoria.

Gracias a lo anterior para llamar a las diferentes variantes se definirán estos parámetros:

- AM-All: aplicacion_BL = 10, prob_aplicacion_BL = 1.0, BL_elitista = false.
- AM-Rand: aplicacion_BL = 10, prob_aplicacion_BL = 0.1, BL_elitista = false.
- AM-Best: aplicacion_BL = 10, prob_aplicacion_BL = 0.1, BL_elitista = true.

5. Procedimiento para desarrollar la práctica.

a. Organización del proyecto.

Para comenzar el desarrollo de la práctica, lo primero fue tomar las decisiones a nivel de estructuras. Con ese fin, se creó la clase Mochila ya explicada.
A continuación se decidió la estructura de ficheros. Usando un fichero para el main, otro para el desarrollo de las funciones de los algoritmos genéticos y otro para los meméticos.
Por último, con el fin de facilitar el uso del software, se desarrolló un Makefile para el compilado automático de las fuentes y diversos bash scripts para generación de múltiples resultados para posteriormente analizarlos.

b. Manual de usuario:

i. Compilación:

Con la orden make.
Se genera un ejecutable llamado QKP.

ii. Ejecución:

Los argumentos del programa son.

1. ./$ejecutable
2. $Algoritmo. Opciones:
    - G para Greedy
    - BL para búsqueda local.
    - AGG-dospuntos. Algoritmo genético generacional con cruce dos puntos.
    - AGG-propuesta. Algoritmo genético generacional con propuesta de cruce.

- AGE-dospuntos. Algoritmo genético estacionario con cruce dos puntos.
- AGE-propuesta. Algoritmo genético estacionario con propuesta de cruce.
- AM-AGG-dospuntos-10,1.0. Algoritmo memético, con genético generacional, aplicación de BL cada 10 generaciones a todos los individuos.
- AM-AGG-dospuntos-10,1.0. Algoritmo memético, con genético generacional, aplicación de BL cada 10 generaciones al 10% de los individuos aleatoriamente.
- AM-AGG-dospuntos-10,1.0mej. Algoritmo memético, con genético generacional, aplicación de BL cada 10 generaciones al 10% de los mejores individuos.

Extra:

- AM-AGG-propuesta-10,1.0. Algoritmo memético, con genético generacional (cruce propuesta), aplicación de BL cada 10 generaciones a todos los individuos.
- AM-AGG-propuesta-10,1.0. Algoritmo memético, con genético generacional (cruce propuesta), aplicación de BL cada 10 generaciones al 10% de los individuos aleatoriamente.
- AM-AGG-propuesta-10,1.0mej. Algoritmo memético, con genético generacional (cruce propuesta), aplicación de BL cada 10 generaciones al 10% de los mejores individuos.
- AM-AGE-dospuntos-10,1.0. Algoritmo memético, con genético estacionario, aplicación de BL cada 10 generaciones a todos los individuos.
- AM-AGE-dospuntos-10,1.0. Algoritmo memético, con genético estacionario, aplicación de BL cada 10 generaciones al 10% de los individuos aleatoriamente.
- AM-AGE-dospuntos-10,1.0mej. Algoritmo memético, con genético estacionario, aplicación de BL cada 10 generaciones al 10% de los mejores individuos.
- AM-AGE-propuesta-10,1.0. Algoritmo memético, con genético estacionario (cruce propuesta), aplicación de BL cada 10 generaciones a todos los individuos.

- AM-AGE-propuesta-10,1.0. Algoritmo memético, con genético estacionario (cruce propuesta), aplicación de BL cada 10 generaciones al 10% de los individuos aleatoriamente.
- AM-AGE-propuesta-10,1.0mej. Algoritmo memético, con genético estacionario (cruce propuesta), aplicación de BL cada 10 generaciones al 10% de los mejores individuos.

3. $fichero de entrada de datos
4. (opcional) $semilla (Para greedy no es necesaria, para bl, si no se indica, no se fijará semilla).

iii. Salida:

Se mostrará por pantalla lo siguiente:

1. W: capacidad de la mochila.
2. Peso de la mochila final.
3. Beneficio final.
4. Asignación (unos y ceros)
5. Última línea: $beneficio $milisegundos

iv. Scripts:

Importante: si se desea utilizar alguno de los scripts se deben comentar todas las salidas de texto por pantalla menos la última línea que saca el beneficio y el tiempo y volver a compilar con make.

Cada uno de esos scripts ejecuta el programa con el algoritmo correspondiente para cada uno de los archivos datos de entrada. Para aquellos con tamaño y densidad comunes, hace la media del beneficio y del tiempo y guarda los resultados en ficheros en formato csv.
Se desarrollaron:

1. ScriptGreedy.sh
2. ScriptBL.sh
3. ScriptAGG-dospuntos.sh
4. ScriptAGG-propuesta.sh
5. ScriptAGE-dospuntos.sh
6. ScriptAGE-propuesta.sh
7. ScriptAM-AGG-dospuntos-10,1.0.sh
8. ScriptAM-AGG-dospuntos-10,0.1.sh
9. ScriptAM-AGG-dospuntos-10,0.1mej.sh

6. Experimentación:

a. Recogida de datos.

Para la recogida de datos experimentales se usaron los scripts mencionados usando la semilla **33**. Posteriormente, se organizaron en tablas.

b. Resultados:

i. AGG-dospuntos:

Resultados AGG-dospuntos			
Tamaño	Densidad	Fitness Medio	Tiempo Medio
100	25	6,13E+04	8,84E+02
100	50	1,25E+05	8,96E+02
100	75	1,83E+05	9,02E+02
100	100	2,47E+05	7,98E+02
200	25	2,33E+05	2,95E+03
200	50	5,36E+05	3,68E+03
200	75	5,06E+05	2,84E+03
200	100	8,76E+05	3,03E+03
300	25	3,99E+05	5,56E+03
300	50	1,07E+06	7,51E+03

ii. AGG-propuesta

Resultados AGG-propuesta			
Tamaño	Densidad	Fitness Medio	Tiempo Medio
100	25	6,18E+04	9,87E+02
100	50	1,29E+05	1,01E+03
100	75	1,88E+05	9,68E+02
100	100	2,54E+05	8,95E+02
200	25	2,39E+05	3,28E+03
200	50	5,54E+05	4,20E+03
200	75	5,40E+05	3,24E+03
200	100	9,04E+05	3,46E+03

300	25	4,17E+05	6,30E+03
300	50	1,12E+06	7,75E+03

iii. AGE-dospuntos

Resultados AGE-dospuntos			
Tamaño	Densidad	Fitness Medio	Tiempo Medio
100	25	5,33E+04	1,01E+03
100	50	1,11E+05	1,03E+03
100	75	1,67E+05	1,04E+03
100	100	2,36E+05	9,83E+02
200	25	2,07E+05	3,05E+03
200	50	4,60E+05	3,34E+03
200	75	4,12E+05	2,76E+03
200	100	7,77E+05	3,05E+03
300	25	3,43E+05	5,34E+03
300	50	9,41E+05	7,28E+03

iv. AGE-propuesta

Resultados AGE-propuesta			
Tamaño	Densidad	Fitness Medio	Tiempo Medio
100	25	6,00E+04	1,16E+03
100	50	1,23E+05	1,15E+03
100	75	1,83E+05	1,21E+03
100	100	2,46E+05	1,06E+03
200	25	2,24E+05	3,20E+03
200	50	5,21E+05	3,88E+03
200	75	4,85E+05	3,21E+03
200	100	8,70E+05	3,46E+03
300	25	3,81E+05	5,83E+03
300	50	1,05E+06	8,18E+03

v. AM-(10,1.0)

Resultados AM-All = AM-AGG-dospuntos-10,1.0			
Tamaño	Densidad	Fitness Medio	Tiempo Medio
100	25	6,16E+04	1,89E+02
100	50	1,25E+05	1,86E+02
100	75	1,83E+05	1,86E+02
100	100	2,48E+05	1,61E+02
200	25	2,36E+05	4,59E+02
200	50	5,30E+05	6,22E+02
200	75	4,96E+05	4,76E+02
200	100	8,76E+05	4,57E+02
300	25	3,98E+05	1,19E+03
300	50	1,07E+06	1,57E+03

vi. AM-(10,0.1)

Resultados AM-Rand = AM-AGG-dospuntos-10,0.1			
Tamaño	Densidad	Fitness Medio	Tiempo Medio
100	25	6,17E+04	5,09E+02
100	50	1,24E+05	5,46E+02
100	75	1,82E+05	5,52E+02
100	100	2,45E+05	4,77E+02
200	25	2,37E+05	1,13E+03
200	50	5,35E+05	1,52E+03
200	75	4,98E+05	1,10E+03
200	100	8,75E+05	1,23E+03
300	25	4,08E+05	1,69E+03
300	50	1,08E+06	2,24E+03

vii. AM-(10,0.1mej)

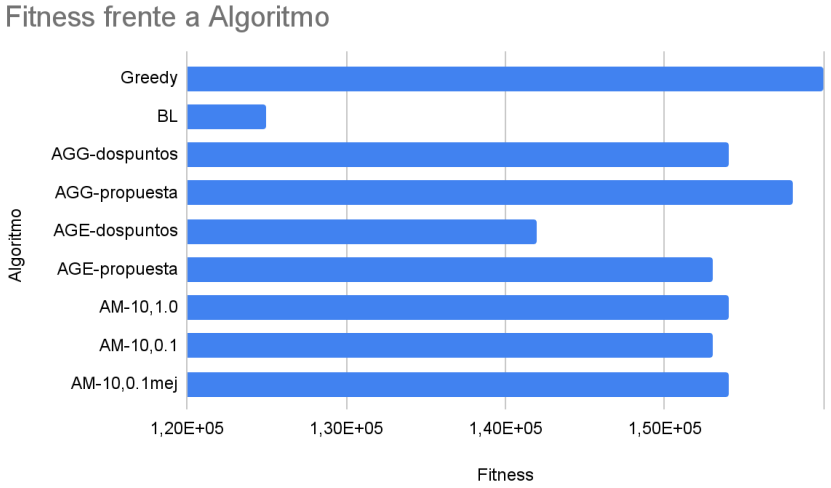
Resultados AM-Best = AM-AGG-dospuntos-10,0.1mej			
Tamaño	Densidad	Fitness Medio	Tiempo Medio
100	25	6,18E+04	4,84E+02
100	50	1,26E+05	5,13E+02
100	75	1,80E+05	5,47E+02
100	100	2,46E+05	4,62E+02

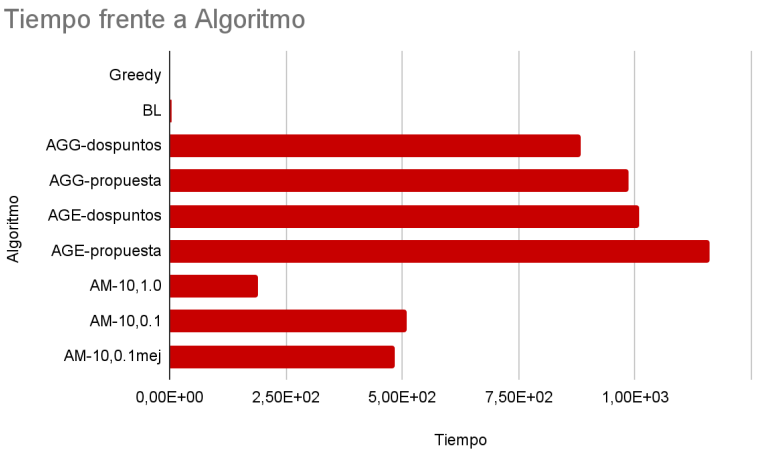
200	25	2,36E+05	1,20E+03
200	50	5,34E+05	1,42E+03
200	75	4,95E+05	1,17E+03
200	100	8,67E+05	1,23E+03
300	25	4,06E+05	1,79E+03
300	50	1,08E+06	2,18E+03

c. Resumen de Resultados:

i. Tamaño 100.

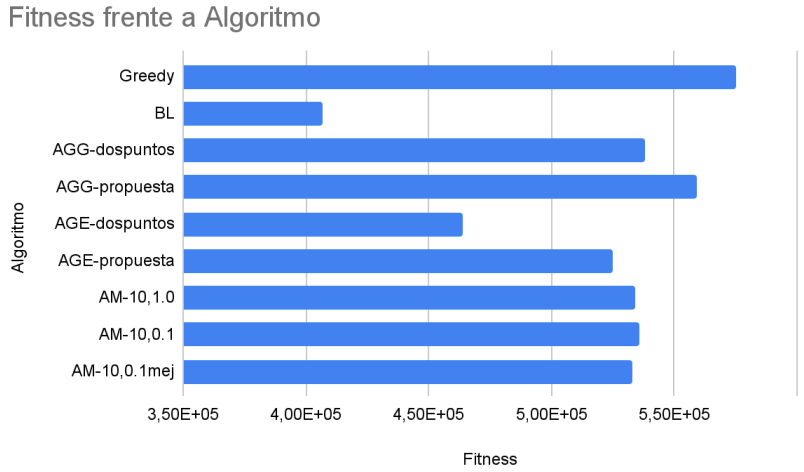
Resumen algoritmos Tamaño 100		
Algoritmo	Fitness	Tiempo
Greedy	1,60E+05	0,00E+00
BL	1,25E+05	5,25E+00
AGG-dospuntos	1,54E+05	8,84E+02
AGG-propuesta	1,58E+05	9,87E+02
AGE-dospuntos	1,42E+05	1,01E+03
AGE-propuesta	1,53E+05	1,16E+03
AM-10,1.0	1,54E+05	1,89E+02
AM-10,0.1	1,53E+05	5,09E+02
AM-10,0.1mej	1,54E+05	4,84E+02

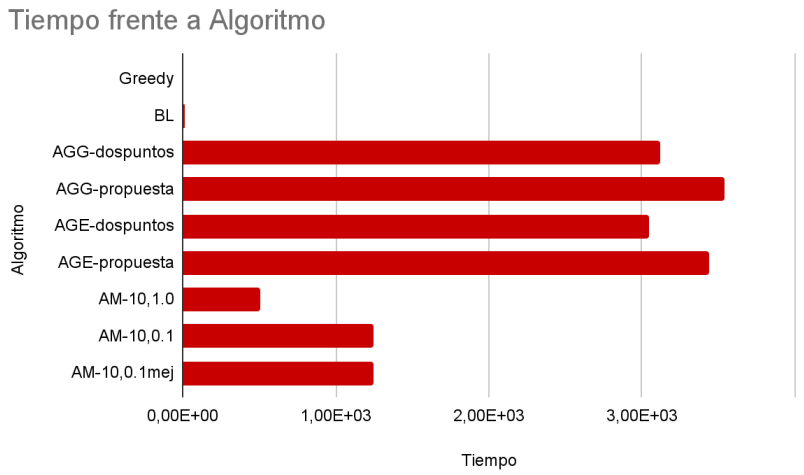




ii. Tamaño 200.

Resumen algoritmos Tamaño 200		
Algoritmo	Fitness	Tiempo
Greedy	5,75E+05	2,00E+00
BL	4,07E+05	1,40E+01
AGG-dospuntos	5,38E+05	3,12E+03
AGG-propuesta	5,59E+05	3,54E+03
AGE-dospuntos	4,64E+05	3,05E+03
AGE-propuesta	5,25E+05	3,44E+03
AM-10,1.0	5,34E+05	5,04E+02
AM-10,0.1	5,36E+05	1,25E+03
AM-10,0.1mej	5,33E+05	1,25E+03

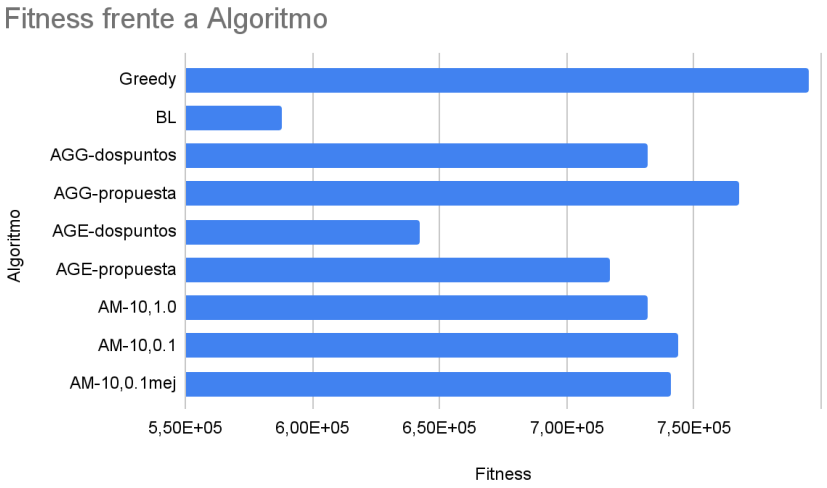


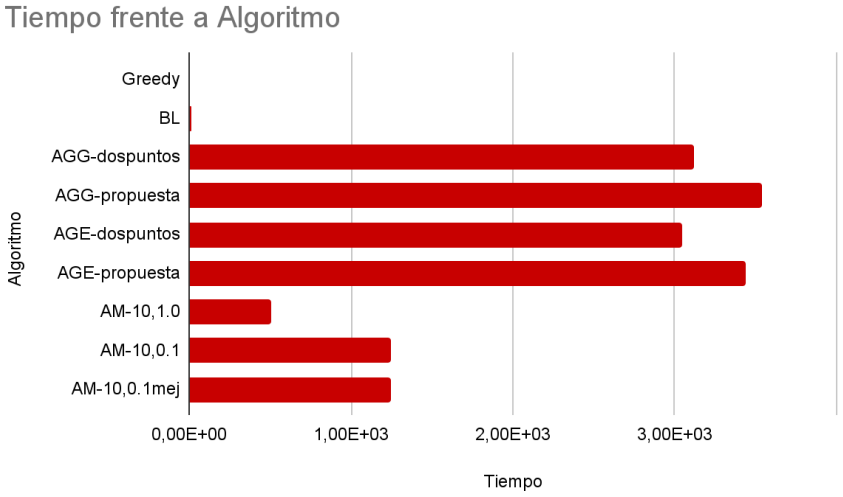


iii. Tamaño 300.

Resumen algoritmos Tamaño 300		
Algoritmo	Fitness	Tiempo
Greedy	7,95E+05	7,50E+00
BL	5,88E+05	5,95E+01
AGG-dospuntos	7,32E+05	6,54E+03
AGG-propuesta	7,68E+05	7,03E+03
AGE-dospuntos	6,42E+05	6,31E+03
AGE-propuesta	7,17E+05	7,01E+03
AM-10,1.0	7,32E+05	1,38E+03
AM-10,0.1	7,44E+05	1,96E+03

AM-10,0.1mej	7,41E+05	1,99E+03





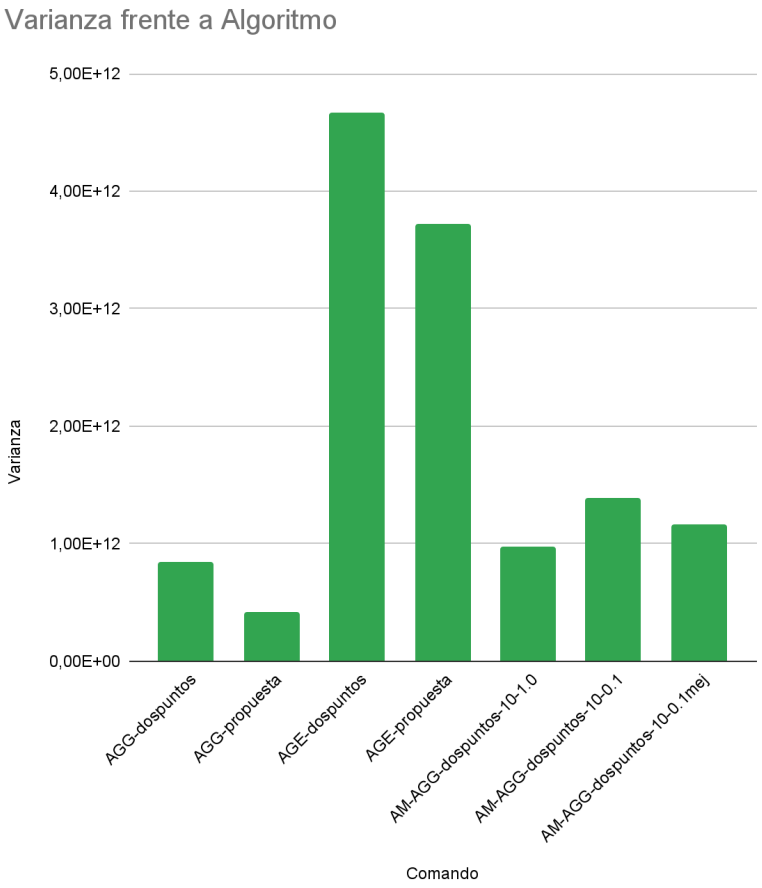
d. Estudio de la varianza de las soluciones para una misma ejecución.

A sabiendas del importante factor estocástico de los algoritmos genéticos y meméticos se decidió hacer una comparativa de varianza entre los diferentes algoritmos para el mismo fichero de entrada con el objetivo de ver hasta que punto pueden ser diferentes los resultados dependiendo del azar para cada algoritmo. Para ello se desarrolló un script que se puede encontrar en el software como EstudioVarianza.sh. Este script ejecuta 100 veces cada algoritmo con el mismo fichero de entrada y sin fijación de semilla por lo que esta será aleatoria, almacena los resultados y calcula la varianza para cada algoritmo. Saca los resultados en un archivo csv resultados_varianza.csv.

Fichero de entrada:
bin/data/jeu_100_25_1.txt

Resultados:

Algoritmo	Fitness	Varianza
AGG-dospuntos	3,23E+04	8,49E+11
AGG-propuesta	3,42E+04	4,21E+11
AGE-dospuntos	1,99E+04	4,67E+12
AGE-propuesta	3,12E+04	3,72E+12
AM-AGG-dospuntos-10-1.0	3,21E+04	9,76E+11
AM-AGG-dospuntos-10-0.1	3,37E+04	1,39E+12
AM-AGG-dospuntos-10-0.1mej	3,21E+04	1,16E+12



Estos resultados son realmente interesantes pues se puede observar que:

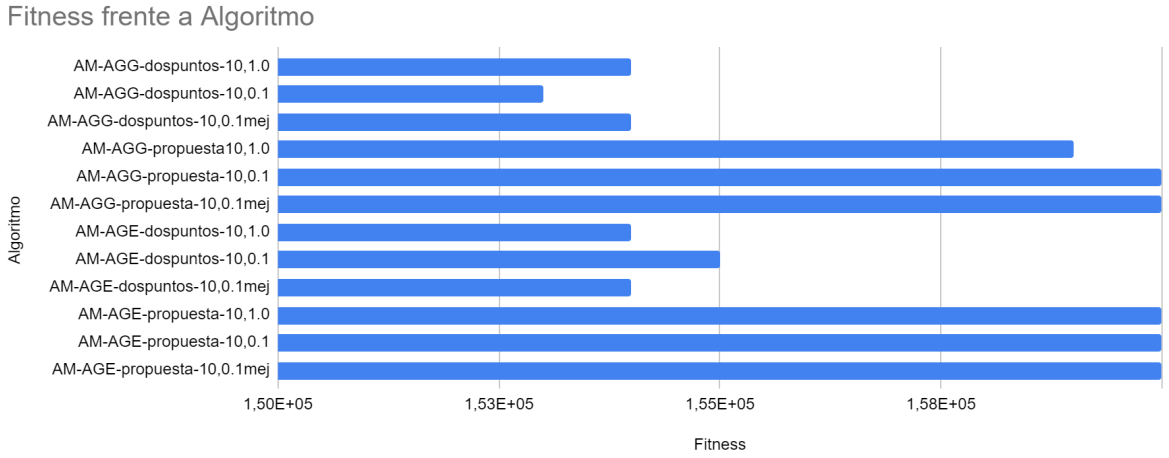
- El algoritmo AGE tiene una varianza muy alta en comparación con el resto de algoritmo.
- Dentro de los genéticos, con la propuesta de cruce se consigue disminuir la varianza respecto el cruce dos puntos.

- Dentro de los meméticos, no encontramos grandes diferencias de varianza entre las variantes, como mucho podemos decir que el que la tiene mayor es el que hace búsqueda local sobre el 10% de los individuos de manera aleatoria.

e. Extra. Experimentación con diferentes variantes de los AM.

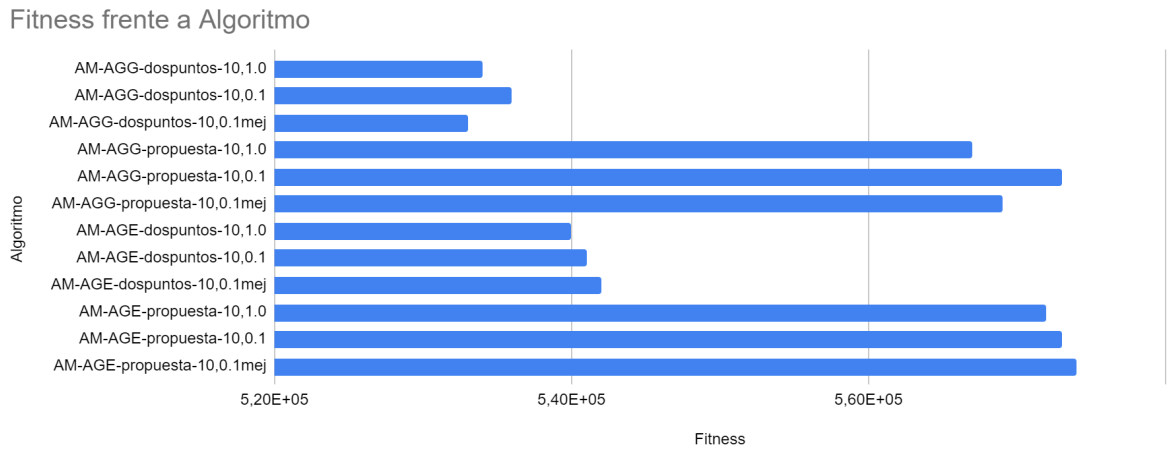
i. Tamaño 100.

Resumen algoritmos Tamaño 100		
Algoritmo	Fitness	Tiempo
AM-AGG-dospuntos-10,1.0	1,54E+05	1,89E+02
AM-AGG-dospuntos-10,0.1	1,53E+05	5,09E+02
AM-AGG-dospuntos-10,0.1mej	1,54E+05	4,84E+02
AM-AGG-propuesta-10,1.0	1,59E+05	1,69E+02
AM-AGG-propuesta-10,0.1	1,60E+05	7,05E+02
AM-AGG-propuesta-10,0.1mej	1,60E+05	5,26E+02
AM-AGE-dospuntos-10,1.0	1,54E+05	1,60E+02
AM-AGE-dospuntos-10,0.1	1,55E+05	5,22E+02
AM-AGE-dospuntos-10,0.1mej	1,54E+05	5,32E+02
AM-AGE-propuesta-10,1.0	1,60E+05	1,87E+02
AM-AGE-propuesta-10,0.1	1,60E+05	7,05E+02
AM-AGE-propuesta-10,0.1mej	1,60E+05	6,60E+02



ii. Tamaño 200.

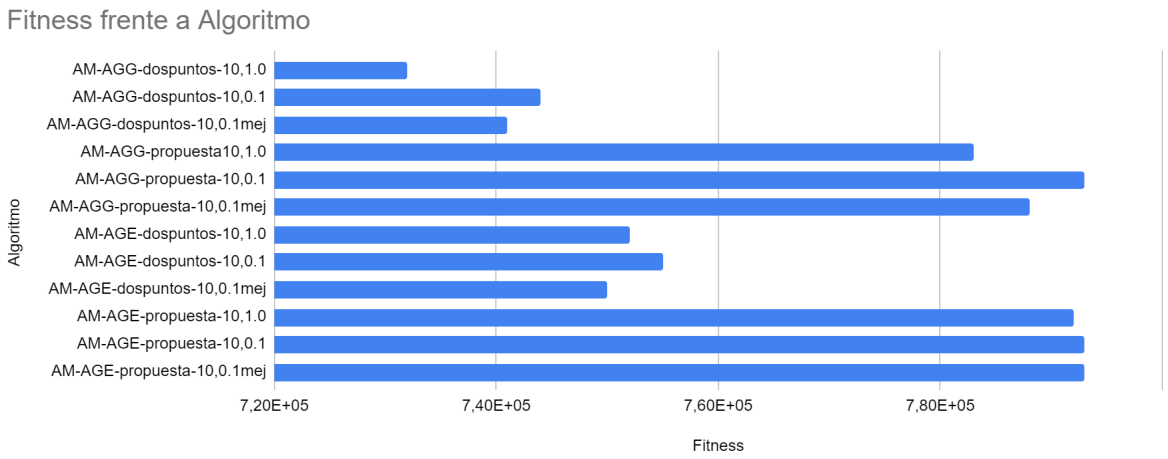
Resumen algoritmos Tamaño 200		
Algoritmo	Fitness	Tiempo
AM-AGG-dospuntos-10,1.0	5,34E+05	5,04E+02
AM-AGG-dospuntos-10,0.1	5,36E+05	1,25E+03
AM-AGG-dospuntos-10,0.1mej	5,33E+05	1,25E+03
AM-AGG-propuesta-10,1.0	5,67E+05	4,66E+02
AM-AGG-propuesta-10,0.1	5,73E+05	1,50E+03
AM-AGG-propuesta-10,0.1mej	5,69E+05	1,29E+03
AM-AGE-dospuntos-10,1.0	5,40E+05	4,30E+02
AM-AGE-dospuntos-10,0.1	5,41E+05	1,13E+03
AM-AGE-dospuntos-10,0.1mej	5,42E+05	1,07E+03
AM-AGE-propuesta-10,1.0	5,72E+05	3,62E+02
AM-AGE-propuesta-10,0.1	5,73E+05	1,50E+03
AM-AGE-propuesta-10,0.1mej	5,74E+05	1,42E+03



iii.Tamaño 300.

Resumen algoritmos Tamaño 300		
Algoritmo	Fitness	Tiempo

AM-AGG-dospuntos-10,1.0	7,32E+05	1,38E+03
AM-AGG-dospuntos-10,0.1	7,44E+05	1,96E+03
AM-AGG-dospuntos-10,0.1mej	7,41E+05	1,99E+03
AM-AGG-propuesta-10,1.0	7,83E+05	1,17E+03
AM-AGG-propuesta-10,0.1	7,93E+05	1,93E+03
AM-AGG-propuesta-10,0.1mej	7,88E+05	2,00E+03
AM-AGE-dospuntos-10,1.0	7,52E+05	1,09E+03
AM-AGE-dospuntos-10,0.1	7,55E+05	1,82E+03
AM-AGE-dospuntos-10,0.1mej	7,50E+05	1,82E+03
AM-AGE-propuesta-10,1.0	7,92E+05	7,76E+02
AM-AGE-propuesta-10,0.1	7,93E+05	1,93E+03
AM-AGE-propuesta-10,0.1mej	7,93E+05	2,06E+03



7. Análisis de resultados:

a. Observaciones Generales.

- El algoritmo Greedy sigue siendo claramente el más competitivo. A nivel de fitness está solo un un poco por delante de los nuevos algoritmos pero a nivel tiempos de ejecución hay una diferencia abismal en favor del greedy.
- Los algoritmos genéticos generacionales suelen ser los que más se acercan al rendimiento de fitness del greedy en comparación con los meméticos, eso sí, a cambio de ser alrededor del doble de lentos que estos últimos.

- Es fácilmente observable que el cruce propuesto aumenta un poco la complejidad cuando se utiliza en los algoritmos genéticos siendo el AGG-propuesta y el AGE-propuesta más lentos que sus respectivos con cruce dos puntos pero a cambio de este aumento, se incrementa el fitness medio, acercándose al del greedy.

b. Genético Generacional vs Genético Estacionario

i. Observaciones de los Resultados:

- Fitness: El AGE tiende a generar soluciones consistentemente peores que el AGG en términos de fitness.
- Tiempo: Aunque los tiempos de ejecución son similares, el AGG puede ser ligeramente más lento, sobre todo evidente en comparaciones específicas como AGG-propuesta y AGE-propuesta.
- Varianza: El AGE tiene una varianza mucho mayor que el AGG.

ii. Análisis de las Diferencias:

- Exploración vs Explotación: El AGG favorece la exploración, mientras que el AGE tiende hacia la explotación. Esto puede explicar la superioridad del AGG en términos de fitness. En el AGE, al solo trabajar con dos individuos por generación, el algoritmo se centra más en encontrar la mejor solución en una espacio de búsqueda más reducido que el AGG. Es por esto que probablemente el AGE se estanque en regiones poco prometedoras.
- Otra consecuencia de lo anterior se ve reflejado en la varianza pues la del AGE es mucho mayor que la del AGG. Esto pasa porque el AGE es mucho más dependiente de recibir una población inicial que englobe un espacio de búsqueda prometedor para que se pueda aprovechar la capacidad de explotación del algoritmo. Sin embargo, en el AGG, como se centra en la exploración, aunque caiga en una zona no tan prometedora, el algoritmo buscará la solución en un área más amplia.

En conclusión, las diferencias entre AGG y AGE en términos de manejo de población y reemplazo influyen significativamente en su desempeño en diferentes contextos de problema. La elección entre ambos depende de la naturaleza específica del problema y de las preferencias del diseñador en términos de exploración y explotación.

c. Cruce dos puntos vs Propuesta de cruce

i. Observaciones de los Resultados:

- Fitness: La propuesta de cruce muestra una mejora significativa en el rendimiento en términos de fitness en comparación con el cruce de dos puntos. Esta mejora es consistente en ambos algoritmos AGG y AGE, lo que sugiere una mayor efectividad de la propuesta de cruce para explorar y explotar el espacio de soluciones.
- Tiempo: Aunque la propuesta de cruce requiere una complejidad computacional ligeramente mayor que el cruce de dos puntos, el aumento en el tiempo de ejecución es asumible. Este costo adicional en tiempo está justificado por el beneficio obtenido en términos de fitness.
- Varianza: La propuesta de cruce reduce notablemente la varianza en comparación con el cruce de dos puntos. Esta reducción en la varianza indica una mayor consistencia en la calidad de las soluciones generadas.

ii. Análisis de las Diferencias:

- Exploración vs Explotación: La propuesta de cruce ofrece una mejor combinación de exploración y explotación en comparación con el cruce de dos puntos. Mientras que el cruce de dos puntos puede limitar la exploración al heredar segmentos genéticos grandes de padres, la propuesta de cruce facilita una combinación más uniforme de información genética de ambos padres, lo que conduce a soluciones más óptimas.
- Equilibrio entre Diversidad y Explotación: La propuesta de cruce mantiene una distribución más uniforme de genes de ambos padres, lo que ayuda a mantener la diversidad en la población y facilita una mejor exploración del espacio de soluciones. Por otro lado, el cruce de dos puntos puede generar descendencia más concentrada alrededor de ciertas regiones del espacio de búsqueda, lo que podría conducir a una convergencia prematura hacia óptimos locales.
- Impacto en la Varianza: La reducción en la varianza observada con la propuesta de cruce sugiere una mayor consistencia en la calidad de las soluciones generadas. Esta mayor consistencia es beneficiosa en términos de estabilidad y confiabilidad del algoritmo.

En resumen, la propuesta de cruce muestra un rendimiento superior en términos de fitness y varianza en comparación con el cruce de dos puntos. Aunque implica una complejidad computacional ligeramente mayor, esta diferencia está justificada por la mejora en la calidad y consistencia de las soluciones generadas.

d. Genéticos vs Meméticos:

i. Observaciones de los Resultados:

- Fitness: Los algoritmos genéticos tienden a encontrar resultados ligeramente mejores en términos de fitness en comparación con los meméticos. Aunque los meméticos logran acercarse significativamente a los genéticos en términos de calidad de soluciones, los genéticos generalmente obtienen resultados ligeramente superiores.
- Tiempo: Los meméticos logran reducir la complejidad computacional, tardando aproximadamente la mitad del tiempo que los algoritmos genéticos en alcanzar resultados comparables en términos de fitness. Esta reducción en el tiempo de ejecución es una ventaja significativa de los meméticos.
- Varianza: Los meméticos muestran una varianza considerablemente mayor en comparación con los algoritmos genéticos. Aunque esto puede indicar una mayor sensibilidad a la configuración de parámetros o a la aleatoriedad inherente al proceso de búsqueda, también puede reflejar una mayor diversidad en las soluciones encontradas.

ii. Análisis de las Diferencias:

- Exploración vs Explotación: Los algoritmos genéticos tienden a enfocarse más en la exploración, explorando ampliamente el espacio de soluciones. Por otro lado, los meméticos combinan la exploración de los algoritmos genéticos con la explotación de la búsqueda local, lo que les permite encontrar soluciones de alta calidad de manera más eficiente.
- Complejidad Computacional: La reducción en la complejidad computacional de los meméticos se debe a su capacidad para aprovechar la eficiencia de la búsqueda local por lo que cuantas más evaluaciones se utilicen en la BL más rápido será el memético.

En resumen, los algoritmos genéticos y meméticos tienen diferencias significativas en términos de enfoque de búsqueda, complejidad computacional y variabilidad en el rendimiento. Mientras que los genéticos tienden a ofrecer resultados ligeramente mejores en términos de fitness, los meméticos logran resultados comparables en menos tiempo. La elección entre ambos dependerá de las necesidades específicas del problema y de las preferencias del diseñador en términos de calidad de soluciones y tiempo de ejecución.

e. Algoritmos Meméticos. Búsqueda Local sobre todas los individuos (1) vs sobre 10% de manera aleatoria(2) vs sobre 10% mejores(3).

i. Observaciones de los Resultados:

- Fitness: La búsqueda local sobre todos los individuos produce resultados de fitness algo peores en comparación con la búsqueda local sobre el 10% aleatorio y el 10% de los mejores individuos.
- Tiempo: La búsqueda local sobre todos los individuos es la más rápida.

ii. Análisis de las Diferencias:

- Eficiencia vs Efectividad: La búsqueda local sobre todos los individuos es eficiente en tiempo pero menos efectiva en la calidad de las soluciones. En contraste, la búsqueda local sobre los mejores individuos logra un equilibrio entre eficiencia y efectividad, obteniendo soluciones competitivas en menos tiempo.
- Aleatoriedad vs Selección: A pesar de la pequeña diferencia en fitness entre las variantes de los meméticos, se observa que el AM-AGG-dospuntos-10,1.0 tiende a mostrar resultados consistentes, mientras que las otras variantes tienden a superarlo a medida que aumenta el tamaño de la mochila. En términos de tiempo, el AM-AGG-dospuntos-10,1.0 sigue siendo el más rápido, lo cual puede atribuirse a la rapidez de las evaluaciones realizadas dentro del algoritmo de búsqueda local. En cuanto a la varianza, no se observan grandes diferencias entre las variantes.

En resumen, los diferentes enfoques de búsqueda local en algoritmos meméticos ofrecen opciones variadas en eficiencia, efectividad y consistencia en la calidad de las soluciones. La elección depende de las necesidades específicas del problema y las preferencias del diseñador en términos de tiempo de ejecución y calidad de las soluciones.

f. Extra. AM-AGG vs AM-AGE.

i. Observaciones de los Resultados:

En general, los resultados entre AM-AGG y AM-AGE no muestran una diferencia significativa en términos de fitness. Sin embargo, se observa que los AM con AGE tienden a producir soluciones ligeramente mejores en algunos casos.

ii. Análisis:

Que el AM con AGE muestre resultados algo mejores que el AM con AGG es del todo curioso si lo ponemos en contexto con el conjunto de resultados de la práctica ya que por sí solos, el AGG era consistentemente superior al AGE. Esto sucede porque el AGE no conseguía resultados igualables al AGG por culpa de su falta de diversidad a medida que avanzaban las generaciones, es decir, la exploración no era suficiente. Sin embargo, en el AM-AGE, al introducir la búsqueda local, aunque la exploración sea la misma, las zonas exploradas se explotan con la búsqueda local haciendo que la combinación exploración explotación sea satisfactoria.

g. Conclusiones:

En el estudio y análisis de los algoritmos genéticos y meméticos aplicados al problema de la optimización de la mochila, se han identificado diversas tendencias y características que proporcionan información valiosa para la selección y diseño de estrategias de resolución eficaces.

Los algoritmos genéticos, tanto generacionales como estacionarios, han demostrado ser eficaces para encontrar soluciones de alta calidad en términos de fitness. Sin embargo, se observa que el enfoque generacional tiende a producir resultados mejores. Este resultado sugiere que para este problema es crucial la exploración del espacio de soluciones antes que la explotación de un área concreta de soluciones.

La implementación de la propuesta de cruce ha mostrado mejoras significativas en el rendimiento en términos de fitness y una reducción notable en la varianza en comparación con el cruce de dos puntos. Aunque implica una mayor complejidad computacional, la propuesta de cruce compensa este costo adicional con la mejora en la calidad y consistencia de las soluciones generadas.

En cuanto a los algoritmos meméticos, se observa que la búsqueda local sobre el 10% de los mejores individuos destaca por su equilibrio entre eficiencia y efectividad, obteniendo soluciones competitivas en menos tiempo al enfocarse en las soluciones más prometedoras. Este enfoque muestra una mayor consistencia y una menor varianza en comparación con la búsqueda local sobre un porcentaje aleatorio de individuos.

En resumen, la elección entre los diferentes enfoques y técnicas de los algoritmos genéticos y meméticos depende de las características específicas del problema, como el tamaño de la instancia, la necesidad de exploración frente a explotación y la disponibilidad de recursos computacionales. La comprensión de estas tendencias y

características es fundamental para diseñar estrategias de optimización efectivas y eficientes para el problema de la optimización de la mochila.

Trabajo realizado por Santiago Herron Mulet