Metaheurísticas: Práctica 3
Búsquedas por Trayectorias
para el Problema de la Máxima Diversidad

David Cabezas Berrido

20079906D

Grupo 2: Viernes

dxabezas@correo.ugr.es

31 de mayo de 2021

Índice

1. Descripción y formulación del problema

Nos enfrentamos al **Problema de la Máxima Diversidad** (**Maximum Diversity Problem, MDP**). El problema consiste en seleccionar un subconjunto m elementos de un conjunto de $n > m$ elementos de forma que se **maximice** la *diversidad* entre los elementos escogidos.

Disponemos de una matriz $D = (d_{ij})$ de dimensión $n \times n$ que contiene las distancias entre los elementos, la entrada (i, j) contiene el valor d_{ij}, que corresponde a la distancia entre el elemento i-ésimo y el j-ésimo. Obviamente, la matriz D es simétrica y con diagonal nula.

Existen distintas formas de medir la diversidad, que originan distintas variantes del problema. En nuestro caso, la diversidad será la suma de las distancias entre cada par de elementos seleccionados.

De manera formal, se puede formular el problema de la siguiente forma:

Maximizar

$$f(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j \tag{1}$$

sujeto a

$$\sum_{i=1}^{n} x_i = m$$
$$x_i = \{0, 1\}, \quad \forall i = 1, \ldots, n.$$

Una solución al problema es un vector binario x que indica qué elementos son seleccionados, seleccionamos el elemento i-ésimo si $x_i = 1$.

Sin embargo, esta formulación es poco eficiente y para la mayoría de algoritmos proporcionaremos otra equivalente pero más eficiente.

El problema es **NP-completo** y el tamaño del espacio de soluciones es $\binom{n}{m}$, de modo que es conveniente recurrir al uso de metaheurísticas para atacarlo.

2. Aplicación de los algoritmos

Los algoritmos para resolver este problema tendrán como entradas la matriz D $(n \times n)$ y el valor m. La salida será un contenedor (vector, conjunto, ...) con los índices de los elementos seleccionados, y no un vector binario como el que utilizamos para la formulación. En nuestro caso (algoritmos implementados en esta práctica) utilizaremos vectores de enteros para representar soluciones.

La evaluación de la calidad de una solución se hará sumando la contribución de cada uno de los elementos, y dividiremos la evaluación en dos funciones. En lugar de calcular la función evaluación como en (1), lo haremos así:

$$f(x) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} d(i,j) = \frac{1}{2}\sum_{i=1}^{m} \text{contrib}(i) \tag{2}$$

La diferencia es que contamos la distancia entre cada dos elementos i, j dos veces, distancia del elemento i-ésimo al j-ésimo y del j-ésimo al i-ésimo. Esto es obviamente más lento que con $j > i$ en la sumatoria, pero nos permite factorizar la evaluación de la solución como suma de las contribuciones de los elementos, lo cuál será útil para reaprovechar cálculos al evaluar soluciones para la Búsqueda Local. Además, representar la solución como un vector de m índices y no un vector binario de longitud n presenta una clara ventaja: las sumatorias van hasta m en lugar de n. No tenemos que computar distancias para luego multiplicarlas por cero como sugería la formulación en (1).

Presentamos el pseudocódigo de la función para calcular la contribución de un elemento x_i.

Algorithm 1: CONTRIB calcula la contribución de un elemento en una solución.

Input: Un vector de índices S.
Input: La matriz de distancias D.
Input: Un entero e correspondiente al índice del elemento.
Output: La contribución del elemento e, como se describe en (2).

```
contrib ← 0
for s in S do
    contrib ← contrib + D[e, s]        // Sumo las distancias del elemento e a cada elemento de S
return contrib
```

Nótese que el elemento e no tiene que pertenecer al conjunto S. Esto obviamente no ocurrirá cuando se vaya a evaluar una solución al completo invocando esta función con la que describiremos a continuación. Pero, de esta forma, permite conocer cómo influirá en la evaluación el añadir un nuevo elemento sin necesidad de añadirlo realmente.

Ahora presentamos el pseudocódigo de la función para evaluar una solución completa.

Algorithm 2: FITNESS calcula la evaluación de una solución.

Input: Un vector de índices S.
Input: La matriz de distancias D.
Output: El valor de la función objetivo sobre la solución compuesta por S, como se describe en (2).

```
fitness ← 0
for e in S do
    fitness ← fitness + contrib(S, D, e)    // Sumo la contribución de cada elemento de la solución
return fitness/2                            // Hemos contado cada distancia dos veces
```

Podemos definir la distancia de un elemento e a un conjunto S como:

$$d(e,S) = \sum_{s \in S} d(e,s) \tag{3}$$

Esta expresión nos será de utilidad para la implementación de los algoritmos.

Gracias a la existencia del Algoritmo 1, podemos obtener esta expresión como $\text{contrib}(S, D, e)$.

Para los pseudocódigos que siguen, suponemos la matriz de distancias D y los parámetros n y m accesibles. El conjunto de todos los elementos es el $\{0, \ldots, n-1\}$, para cuando nos refiramos a elementos de fuera de un subconjunto de ellos.

En esta práctica implementaremos 4 algoritmos basados en trayectorias, y los compararemos entre sí y con la búsqueda local y el greedy de la práctica 1. Adicionalmente, compararemos para el Enfriamiento Simulado el esquema de Cauchy Modificado con el enfriamiento proporcional, también en la hibridación con ILS.

Todos los algoritmos de esta práctica parten de soluciones aleatorias. La siguiente función permite construir las soluciones aleatorias que utilizaremos como partida.

Algorithm 3: RandomSol proporciona una solución válida aleatoria

Output: Una solución válida del MDP obtenida aleatoriamente.
$E \leftarrow \{0, \ldots, n-1\}$ `// Vector con todos los elementos.`
shuffle(E)
$S \leftarrow \emptyset$ `// La solución empieza vacía.`
while $|S| < m$ **do**
  $S \leftarrow S \cup \{E[|S|]\}$ `// Seleccionamos los` m `primeros elementos de` E`, que son aleatorios.`
return S

La mayoría de algoritmos implementados en esta práctica hacen uso de la búsqueda local (con primer mejor). El siguiente algoritmo implementa esta búsqueda. Supondremos declaradas las variables globales $LIMIT$, que valdrá 100000 o 10000 dependiendo de si el problema es de trayectoria simple o múltiple; y $EVALS$, las evaluaciones hasta el momento, esta variable comienza a 0 y sólo es reseteada en los problemas de trayectorias múltiples (donde $LIMIT$ vale 10000), y se resetea 10 veces para que el total de evaluaciones siempre sea 100000.

Algorithm 4: LocalSearch modifica una solución con varias iteraciones de búsqueda local con primer mejor.

Input: Solución de partida S.
Output: La solución S se modifica (no se devuelve) con varias iteraciones de búsqueda local.
$E \leftarrow \{0, \ldots, n-1\}$ `// Vector con todos los elementos.`
$carryon \leftarrow true$
while $carryon$ **do**
  $carryon \leftarrow false$
  $lowest \leftarrow$ indice del elemento de S que menos contribuye, minimiza contrib$(S, D, S[lowest])$
  $min\_contrib \leftarrow$ contrib$(S, D, lowest)$ $E \leftarrow$ shuffle(E) `// Para explorar los posibles vecinos en orden aleatorio.`
  for e ***in*** E **do**
    if $e \in S$ **then**
      continue `// Si ya está escogido, no lo cuento.`
    $contrib \leftarrow$ contrib$(S, D, e) - D[e, S[lowest]]$ `// Contribución a la solución sin el elemento a sustituir.`
    $EVALS \leftarrow EVALS + 1$ `// He evaludado una posible solución.`
    if $contrib > min\_contrib$ **then**
      $S.fitness \leftarrow S.fitness + contrib - min\_contrib$ `// Fitness de la nueva solución`
      $carryon \leftarrow true$ `// Toca saltar, lo que completa la iteración`
      $S[lowest] \leftarrow e$ `// Saltamos a la nueva solución`
    if $carryon == true$ *or* $EVALS \geq LIMIT$ **then**
      break `// Se cumple alguna de las condiciones de parada`

3. Descripción de los algoritmos

3.1. Enfriamiento Simulado

Para implementar el algoritmo de Enfriamiento Simulado, hemos separado los operadores de generación de vecino y salto.

Algorithm 5: MUTATE genera un vecino aleatorio en un entorno de la solución y calcula la diferencia de fitness.

Input: Solución de partida S.
Output: Índice del elemento a eliminar.
Output: Elemento a añadir.
Output: Diferencia de fitness entre el vecino y la solución S.
$index\_out \leftarrow$ entero aleatorio entre 0 y $|S| - 1$
$elem\_in \leftarrow$ elemento aleatorio entre 0 y $n - 1$
while $elem\_in \in S$ **do**
  $elem\_in \leftarrow$ elemento aleatorio entre 0 y $n - 1$
$contrib\_in \leftarrow \text{contrib}(S, D, elem\_in) - D[elem\_in, S[index\_out]]$ `// Contribución del nuevo elemento, no hay que contar su distancia al que vamos a quitar.`
$contrib\_out \leftarrow \text{contrib}(S, D, S[index\_out])$
$delta \leftarrow contrib\_in - contrib\_out$
$EVALS \leftarrow EVALS + 1$
return $elem\_in$
return $index\_out$
return $delta$

Algorithm 6: JUMP desplaza la solución al vecino indicado.

Input: Solución de partida S.
Input: Índice del elemento a eliminar: $index\_out$.
Input: Elemento a añadir: $elem\_in$.
Input: Diferencia de fitness entre el vecino y la solución S: $delta$.
Output: Nueva solución (no devuelve nada, modifica S).
$S[index\_out] \leftarrow elem\_in$
$S.fitness \leftarrow S.fitness + delta$

En este caso, el número de evaluaciones no se se resetea, y el límite se fija en 100000. El cuerpo del algoritmo queda de la siguiente forma.

Algorithm 7: EnfriamientoSimulado

Output: Una solución factible obtenida por enfriamiento simulado con Cauchy Modificado.

$exitos \leftarrow 1$ `// Para que entre la primera vez en el bucle`
$max\_vecinos \leftarrow 10n$
máx $\_exitos \leftarrow 0.1 * max\_vecinos$
$best\_fitness \leftarrow 0$
$M \leftarrow LIMIT/max\_vecinos$
$S \leftarrow$ randomSol
evaluate(S) `// Asigna` $S.fitness \leftarrow$ fitness(S).
$EVALS \leftarrow EVALS + 1$
$mu \leftarrow 0.3$
$phi \leftarrow 0.3$
$T \leftarrow -mu \cdot S.fitness/\log phi$ `// Temperatura inicial`
$T_f \leftarrow 10^{-3}$
while $T < T_f$ **do**
  $T_f \leftarrow T_f/10$ `// La temperatura final debe ser menor que la inicial`
$beta \leftarrow (T - T_f)/(M \cdot T \cdot T_f)$
while $exitos > 0$ *and* $EVALS < LIMIT$ **do**
  $exitos \leftarrow 0$
  $vecinos \leftarrow 0$
  while $vecinos < max\_vecinos$ *and* $exitos < max\_exitos$ **do**
    $index\_out, elem\_in, delta \leftarrow$ mutate(S)
    $vecinos \leftarrow vecinos + 1$
    $x \leftarrow$ número aleatorio en una uniforme $[0, 1]$
    if $delta > 0$ *or* $x \leq \exp(delta/T)$ **then**
      jump($S, index\_out, elem\_in, delta$)
      $exitos \leftarrow exitos + 1$
      if $S.fitness > best\_fitness$ **then**
        $best\_sol \leftarrow S$ `// En realidad sólo guardo la fitness, no la solución`
        $best\_fitness \leftarrow S.fitness$
  $T \leftarrow T/(1 + beta \cdot T)$ `// Enfriamiento`
return $best\_sol$

También incorporamos una comparación con el esquema de enfriamiento proporcional. La única diferencia es que el enfriamiento pasa a ser $T \leftarrow 0.9 \cdot T$, y sobran M y $beta$.

3.2. Búsqueda Multiarranque Básica

Este algoritmo se limita a ejecutar 10 búsquedas locales, cada una con 10000 evaluaciones ($LIMIT = 10000$ y $EVALS$ se resetea a 0 para cada búsqueda), al final nos quedamos con la mejor de las 10 soluciones alcanzadas.

Algorithm 8: BMB

Output: Una solución factible obtenida por Búsqueda Multiarranque Básica.

$best\_fitness \leftarrow 0$
for $i = 0, \ldots, 9$ **do**
  $EVALS \leftarrow 0$
  $S \leftarrow$ randomSol
  evaluate(S) `// Asigna` $S.fitness \leftarrow$ fitness(S).
  $EVALS \leftarrow EVALS + 1$
  localSearch(S)
  if $S.fitness > best\_fitness$ **then**
    $best\_sol \leftarrow S$ `// En realidad sólo guardo la fitness, no la solución`
    $best\_fitness \leftarrow S.fitness$
return $best\_sol$

3.3. Búsqueda Local Reiterada

Para este algoritmo, necesitamos el siguiente operador de mutación. Se eliminan aleatoriamente t elementos de S (los t primeros tras barajar) y se sustituyen por t elementos de fuera elegidos aleatoriamente.

Algorithm 9: MUTATE modifica la solución cambiando el 10 % de los elementos seleccionados por elementos aleatorios de fuera.

Input: Solución de partida S.
Input: Número de elementos a modificar: t.
Output: Modifica (no devuelve) t elementos aleatorios de S.

```
E ← {0, ..., n − 1}
shuffle(S)
shuffle(E)
new ← ∅
j ← 0
while |new| < t do
    if E[j] ∉ S then
        new ← new ∪ {E[j]}
    j ← j + 1
for j = 0, ..., |new| − 1 do
    S[j] ← new[j]
evaluate(S)
EVALS ← EVALS + 1
```

El algoritmo ILS es similar al anterior, la diferencia es que cada solución parte de una mutación aleatoria de la mejor encontrada. El límite de ejecucioes es 10000, y se resetea el número de evaluaciones 10 veces.

Algorithm 10: ILS

Output: Una solución factible obtenida por Búsqueda Local Reiterada.

```
t ← 0.1 · m
best_sol ← randomSol
evaluate(best_sol)
best_fitness ← 0
for i = 0, ..., 9 do
    EVALS ← 0
    S ← best_sol
    mutate(S, t)
    localSearch(S)
    if S.fitness > best_fitness then
        best_sol ← S
        best_fitness ← S.fitness
return best_sol
```

Esta implementación realiza 100001 iteraciones en lugar de 100000, e igual le ocurre a la siguiente variante. Aunque esto no es relevante para el comportamiento del algoritmo.

3.4. Híbridación ILS-ES

Este algoritmo es idéntico al anterior, con la diferencia de que utiliza Enfriamiento Simulado en lugar de Búsqueda Local. Para ello, necesitamos una variante del Algoritmo 7 que modifique una solución de entrada en lugar de generar una nueva.

Algorithm 11: EnfriamientoSimulado

Input: Una solución factible de partida: S.
Output: La solución S modificada con varias iteraciones de Enfriamiento Simulado con Cauchy Modificado.

$exitos \leftarrow 1$ // Para que entre la primera vez en el bucle
$max\_vecinos \leftarrow 10n$
máx $\_exitos \leftarrow 0.1 * max\_vecinos$
$M \leftarrow LIMIT/max\_vecinos$
$EVALS \leftarrow EVALS + 1$
$mu \leftarrow 0.3$
$phi \leftarrow 0.3$
$T \leftarrow -mu \cdot S.fitness/\log phi$ // Temperatura inicial
$T_f \leftarrow 10^{-3}$
while $T < T_f$ **do**
  $T_f \leftarrow T_f/10$ // La temperatura final debe ser menor que la inicial
$beta \leftarrow (T - T_f)/(M \cdot T \cdot T_f)$
while $exitos > 0$ *and* $EVALS < LIMIT$ **do**
  $exitos \leftarrow 0$
  $vecinos \leftarrow 0$
  while $vecinos < max\_vecinos$ *and* $exitos < max\_exitos$ **do**
    $index\_out, elem\_in, delta \leftarrow$ mutate(S)
    $vecinos \leftarrow vecinos + 1$
    $x \leftarrow$ número aleatorio en una uniforme $[0, 1]$
    if $delta > 0$ *or* $x \leq \exp(delta/T)$ **then**
      jump$(S, index_out, elem_in, delta)$
      $exitos \leftarrow exitos + 1$
  $T \leftarrow T/(1 + beta \cdot T)$ // Enfriamiento
return $best\_sol$

Con esta nueva implementación, el cuerpo del algoritmo ILS-ES es exactamente igual al de ILS (Algoritmo 10) pero cambiando localSearch(S) por enfriamientoSimulado(S).

Hay que tener cuidado, ya que hemos usado un nombre confuso para algunas funciones. El operador Mutate llamado por el Enfriamiento Simulado busca un vecino y calcula el cambio de fitness, corresponde al Algoritmo 5. Sin embargo, el operador Mutate invocado por ILS corresponde al Algoritmo 9.

También incluimos la comparación con el modelo de enfriamiento proporcional. Como el número de enfriamientos va a ser es mucho menor, en lugar de 0.9 usamos 0.5: $T \leftarrow 0.5 \cdot T$. Esto quizá enfríe demasiado rápido.

3.5. Búsqueda Local

Procedemos con la descripción del algoritmo de Búsqueda Local que se nos ha presentado en el seminario. Este algoritmo utiliza la técnica del Primer Mejor, en la que se van generando soluciones en el entorno de la actual y se salta a la primera con mejor evaluación. Para la implementación del algoritmo, necesitamos distintos elementos.

Este algoritmo se implementó en la práctica 1, y utiliza conjuntos de enteros en lugar de vectores para representar las soluciones.

El primer elemento, es una función para generar una solución aleatoria de partida. Simplemente se eligen m elementos diferentes del conjunto. Por comodidad, también calculamos el complementario.

Algorithm 12: RandomSol proporciona una solución válida aleatoria

Input: El entero m.
Input: El entero n.
Output: Una solución válida del MDP obtenida aleatoriamente.
Output: El complementario de la solución obtenida.

```
E ← {0, ..., n − 1}                          // Conjunto con los elementos no seleccionados
S ← ∅                                        // La solución empieza vacía
while |S| < m do
    e ← elemento aleatorio de E
    E ← E\{e}
    S ← S ∪ {e}
return S
return E                                     // El complementario
```

Lo siguiente que necesitamos es un método para generar las soluciones del entorno. Estas soluciones se consiguen sustituyendo el menor contribuyente de la solución actual por otro candidato. Presentamos el código para obtener el menor contribuyente.

Algorithm 13: lowestContrib obtiene el elemento de S que menos contribuye en la valoración.

Input: Un conjunto de elementos S.
Input: La matriz de distancias D.
Output: El elemento de S que minimiza contrib(S, S, e) con $e \in S$.
Output: Su contribución, para la factorización de la función objetivo.

```
lowest ← primer elemento de S
min_contrib ← contrib(S, D, lowest)
for s in S do
    contrib ← contrib(S, D, s)
    if contrib < min_contrib then
        min_contrib ← contrib
        lowest ← s                  // Si encuentro un candidato con menor contribución, actualizo
return lowest
return min_contrib
```

En el caso de que S se represente como un conjunto, no sabemos cuál será el primer elemento (depende de la implementación del iterador). Pero esto no es relevante, ya que vale cualquier elemento de S.

Finalmente, proporcionamos el algoritmo de Búsqueda Local para actualizar la solución por otra del entorno iterativamente hasta encontrar un máximo local (una solución mejor que todas las de su entorno) o llegar a un límite de evaluaciones de la función objetivo: $LIMIT = 100000$. Las soluciones del entorno se generan aleatoriamente.

Algorithm 14: LocalSearch

Input: El entero m.
Input: La matriz de distancias D, $n \times n$.
Output: Una solución válida del MDP por el algoritmo de BS que hemos descrito, junto con su evaluación.

```
S ← randomSol(m, n)                          // Comenzamos con una solución aleatoria
E ← {0, ..., n − 1}\S                        // randomSol también devuelve el complementario
fitness ← fitness(S)                         // Diversidad de la solución
E ← vector(E)                                // No importa el orden, pero debe poder barajarse
carryon ← true
LIMIT ← 100000                               // Límite de llamadas a la función de evaluación
CALLS ← 0
while carryon do
    carryon ← false
    lowest = lowestContributor(S, D)
    min_contrib ← contrib(S, D, lowest)      // Se calcula dentro de lowestContributor
    S ← S\{lowest}
    E ← shuffle(E)
    for e in E do
        contrib ← contrib(S, D, e)
        CALLS ← CALLS + 1                    // He evaludado una posible solución
        if contrib > min_contrib then
            fitness ← fitness + contrib − min_contrib   // Diversidad de la nueva solución
            carryon ← true                   // Toca saltar, lo que completa la iteración
            S ← S ∪ {e}                      // Saltamos a la nueva solución
            E ← E\{e}
            E ← E ∪ {lowest}
        if carryon == true or CALLS ≥ LIMIT then
            break                            // Se cumple alguna de las condiciones de parada
if |S| < m then
    S ← S ∪ {lowest}     // Si salimos porque no encontramos una mejor, recuperamos la solución
return S
return fitness
```

Cabe destacar que en este algoritmo se calcula la fitness factorizando. Esto acelera mucho los cálculos, ya que hay que evaluar muchas soluciones diferentes.

4. Algoritmo de comparación: Greedy

Para comparar la eficacia de cada algoritmos, lo compararemos con el algoritmo **Greedy**. El algoritmo consiste en empezar por el elemento más lejano al resto e ir añadiendo el elemento que más contribuya hasta completar una solución válida.

Este algoritmo también se implementó en la primera práctica y utiliza un conjunto de enteros.

Como elemento más lejano al resto se toma el elemento cuya suma de las distancias al resto sea la mayor. Y en cada iteración se introduce el elemento cuya suma de las distancias a los seleccionados sea mayor. Es decir, utilizamos la definición de (3).

Para calcular ambos valores, usamos la siguiente función, que permite obtener el de entre un conjunto de candidatos más lejano (en el sentido que acabamos de comentar) a los elementos de un conjunto dado. El código para calcularlo es similar al del algoritmo **??**.

Algorithm 15: FARTHEST obtiene el candidato más lejano a los elementos de S.

Input: Un conjunto de candidatos C.
Input: Un conjunto de elementos S.
Input: La matriz de distancias D.
Output: El candidato más lejano en el sentido de (3).

```
farthest ← primer elemento de C
max_contrib ← contrib(S, D, farthest)
for e in C do
    contrib ← contrib(S, D, e)
    if contrib > max_contrib then
        max_contrib ← contrib
        farthest ← e                    // Si encuentro un candidato con mayor contribución, actualizo
return farthest
```

En el caso de que C se represente como un conjunto, no sabemos cuál será el primer elemento (depende de la implementación del iterador). Pero esto no es relevante, ya que vale cualquier elemento de C.

Ya estamos en condiciones de proporcionar una descripción del algoritmo Greedy.

Algorithm 16: GREEDY

Input: La matriz de distancias D.
Input: El entero m.
Output: Una solución válida del MDP obtenida como hemos descrito anteriormente, y su diversidad.

```
C ← {0, ..., n − 1}                          // En principio los n elementos son candidatos
S ← ∅                                          // La solución empieza vacía
farthest ← farthest(C, C, D)                   // Elemento más lejano al resto
C ← C\{farthest}
S ← S ∪ {farthest}
while |S| < m do
    farthest ← farthest(C, S, D)               // Elemento más lejano a los seleccionados
    C ← C\{farthest}
    S ← S ∪ {farthest}
return S
return fitness(S)
```

5. Desarrollo de la práctica

La implementación de los algoritmos y la experimentación con los mismos se ha llevado acabo de C++, utilizando la librería STL. Para representar la soluciones hemos hecho uso del tipo `vector`.

La mayoría de operadores (mutación, generación de vecino, salto, búsqueda local) se implementan como métodos de una clase Solucion.

En el enfriamiento simulado, se utilizan las funciones exponencial y logaritmo neperiano de la biblioteca `math.h`.

Para medir los tiempos de ejecución se utiliza la función `clock` de la librería `time.h`.

A lo largo de la práctica se utilizan acciones aleatorias. Utilizamos la librería `stdlib.h` para la generación de enteros (no negativos) pseudoaleatorios con `rand` y fijamos la semilla con `srand`. Se barajan vectores con la función `random_shuffle` de la librería `algorithm`.

Para las acciones que se realizan con cierta probabilidad, es necesario generar flotantes pseudoaleatorios en el intervalos $[0, 1]$. Para esto, se genera un entero no negativo con `rand` y se divide entre el máximo posible (RAND_MAX).

Se almacena la matriz de distancias completa (no sólo un triángulo) por comodidad de los cálculos.

Se utiliza optimización de código `-O2` al compilar.

5.1. Manual de usuario

A continuación detallamos instrucciones para lanzar los ejecutables.

Tenemos los siguientes ejecutables:

- **ES:** Enfriamiento Simulado con Cauchy Modificado.
- **ES-proporcional:** Enfriamiento Simulado con enfriamiento proporcional.
- **BMB:** Búsqueda Multiarranque Básica.
- **ILS:** Búsqueda Local Reiterada.
- **ILS-ES:** Hibridación de ILS y Enfriamiento Simulado con Cauchy Modificado.
- **ILS-ES-proporcional:** Hibridación de ILS y Enfriamiento Simulado con enfriamiento proporcional.

Todos ellos devuelven la evaluación de la solución obtenida y el tiempo de ejecución por salida estándar. Leen el fichero por entrada estándar, así que es conveniente redirigirla. Todos los archivos reciben la semilla como parámetro.

Además, todos los archivos de búsqueda local reciben la semilla como parámetro. Ejemplo:

```
bin/ES 197 < datos/MDG-a_1_n500_m50.txt >> salidas/ES.txt
```

En la carpeta **software** se incluye el script usado para lanzar todas las ejecuciones, `run.sh`. También se incluye el `Makefile` que compila los ejecutables.

6. Experimentación y análisis

Toda la experimentación se realiza en mi ordenador portátil personal, que tiene las siguientes especificaciones:

- OS: Ubuntu 20.04.2 LTS x86_64.
- RAM: 8GB, DDR4.
- CPU: Intel Core i7-6700HQ, 2.60Hz.

6.1. Casos de estudio y resultados

Tratamos varios casos con distintos parámetros n y m. En cada caso se utiliza una semilla diferente, pero se usa la misma para todos los algoritmos. A continuación presentamos una tabla con los casos estudiados. Para cada caso indicamos los valores de n y m y la semilla que se utiliza.

Caso	n	m	Seed
MDG-a_10_n500_m50	500	50	13
MDG-a_1_n500_m50	500	50	19
MDG-a_2_n500_m50	500	50	25
MDG-a_3_n500_m50	500	50	31
MDG-a_4_n500_m50	500	50	37
MDG-a_5_n500_m50	500	50	43
MDG-a_6_n500_m50	500	50	49
MDG-a_7_n500_m50	500	50	55
MDG-a_8_n500_m50	500	50	61
MDG-a_9_n500_m50	500	50	67
MDG-b_21_n2000_m200	2000	200	73
MDG-b_22_n2000_m200	2000	200	79
MDG-b_23_n2000_m200	2000	200	85
MDG-b_24_n2000_m200	2000	200	91
MDG-b_25_n2000_m200	2000	200	97
MDG-b_26_n2000_m200	2000	200	103
MDG-b_27_n2000_m200	2000	200	109
MDG-b_28_n2000_m200	2000	200	115
MDG-b_29_n2000_m200	2000	200	121
MDG-b_30_n2000_m200	2000	200	127
MDG-c_10_n3000_m400	3000	400	133
MDG-c_13_n3000_m500	3000	500	139
MDG-c_14_n3000_m500	3000	500	145
MDG-c_15_n3000_m500	3000	500	151
MDG-c_19_n3000_m600	3000	600	157
MDG-c_1_n3000_m300	3000	300	163
MDG-c_20_n3000_m600	3000	600	169
MDG-c_2_n3000_m300	3000	300	175
MDG-c_8_n3000_m400	3000	400	181
MDG-c_9_n3000_m400	3000	400	187

Tabla 1: Tabla con los parámetros y semillas de cada caso. Ordenando los nombres de los ficheros por orden alfabético (el orden en el que los procesa el script), las semillas son números del 13 al 187 saltando de 6 en 6.

Ahora mostraremos para cada algoritmo una tabla con los estadísticos (Desviación y Tiempo) que han obtenido en cada caso.

Greedy

Caso	Coste obtenido	Desv	Tiempo (s)
MDG-a_1_n500_m50	7610.42	2.85	0.001375
MDG-a_2_n500_m50	7574.39	2.54	0.001293
MDG-a_3_n500_m50	7535.96	2.88	0.001304
MDG-a_4_n500_m50	7551.52	2.81	0.001281
MDG-a_5_n500_m50	7540.14	2.77	0.001284
MDG-a_6_n500_m50	7623.65	1.93	0.001278
MDG-a_7_n500_m50	7594.62	2.28	0.0014
MDG-a_8_n500_m50	7625.94	1.61	0.001367
MDG-a_9_n500_m50	7547.25	2.87	0.001351
MDG-a_10_n500_m50	7642.27	1.77	0.001893
MDG-b_21_n2000_m200	11099332.620328	1.77	0.319017
MDG-b_22_n2000_m200	11149879.733826	1.21	0.313017
MDG-b_23_n2000_m200	11119613.974858	1.6	0.303374
MDG-b_24_n2000_m200	11106996.970212	1.63	0.311278
MDG-b_25_n2000_m200	11114220.292214	1.61	0.306411
MDG-b_26_n2000_m200	11132801.799043	1.41	0.306542
MDG-b_27_n2000_m200	11130608.965587	1.55	0.310595
MDG-b_28_n2000_m200	11110673.520354	1.5	0.318429
MDG-b_29_n2000_m200	11156328.082493	1.25	0.306362
MDG-b_30_n2000_m200	11109767.818822	1.65	0.296905
MDG-c_1_n3000_m300	24617010	1.07	1.501668
MDG-c_2_n3000_m300	24547293	1.44	1.464132
MDG-c_8_n3000_m400	43056071	0.88	2.546235
MDG-c_9_n3000_m400	42958639	1.1	2.569214
MDG-c_10_n3000_m400	42959794	1.19	2.566065
MDG-c_13_n3000_m500	66493045	0.78	3.67213
MDG-c_14_n3000_m500	66449858	0.79	3.767131
MDG-c_15_n3000_m500	66468837	0.78	3.78725
MDG-c_19_n3000_m600	94929882	0.74	5.183856
MDG-c_20_n3000_m600	94979205	0.69	5.582157

Tabla 2: Evaluación de las soluciones y estadísticos *Desv* y *Tiempo* obtenidos por el algoritmo Greedy en cada caso de estudio.

Media de los estadísticos:

Desv	Tiempo (s)
1.63	1.19

Búsqueda Local

Caso	Coste obtenido	Desv	Tiempo (s)
MDG-a_1_n500_m50	7623.23	2.69	0.001809
MDG-a_2_n500_m50	7590.18	2.34	0.001391
MDG-a_3_n500_m50	7544.94	2.76	0.001204
MDG-a_4_n500_m50	7576.44	2.49	0.0012
MDG-a_5_n500_m50	7484.27	3.49	0.001308
MDG-a_6_n500_m50	7570.96	2.61	0.001297
MDG-a_7_n500_m50	7654.98	1.5	0.001608
MDG-a_8_n500_m50	7623.78	1.64	0.002379
MDG-a_9_n500_m50	7612.74	2.02	0.001494
MDG-a_10_n500_m50	7619.52	2.07	0.001959
MDG-b_21_n2000_m200	11181874.0007	1.04	0.099777
MDG-b_22_n2000_m200	11167876.184	1.05	0.092492
MDG-b_23_n2000_m200	11176568.0611	1.09	0.107634
MDG-b_24_n2000_m200	11188223.318	0.91	0.107425
MDG-b_25_n2000_m200	11181859.8196	1.01	0.090053
MDG-b_26_n2000_m200	11193478.832	0.88	0.122694
MDG-b_27_n2000_m200	11211629.6839	0.83	0.112468
MDG-b_28_n2000_m200	11151089.4629	1.14	0.079449
MDG-b_29_n2000_m200	11183039.6644	1.01	0.09833
MDG-b_30_n2000_m200	11159590.8213	1.21	0.090033
MDG-c_1_n3000_m300	24729057	0.62	0.601221
MDG-c_2_n3000_m300	24738675	0.67	0.584432
MDG-c_8_n3000_m400	43200330	0.55	1.264437
MDG-c_9_n3000_m400	43157977	0.64	1.241837
MDG-c_10_n3000_m400	43188306	0.66	1.195051
MDG-c_13_n3000_m500	66636142	0.56	2.304507
MDG-c_14_n3000_m500	66727635	0.38	2.430114
MDG-c_15_n3000_m500	66808383	0.28	2.78715
MDG-c_19_n3000_m600	95244690	0.41	3.572005
MDG-c_20_n3000_m600	95324379	0.33	3.598978

Tabla 3: Evaluación de las soluciones y estadísticos *Desv* y *Tiempo* obtenidos por el algoritmo de Búsqueda Local con Primer Mejor en cada caso de estudio.

Media de los estadísticos:

Desv	Tiempo (s)
1.3	0.69

Enfriamiento Simulado (Cauchy Modificado)

Caso	Coste obtenido	Desv	Tiempo (s)
MDG-a_1_n500_m50	7670.07	2.09	0.006072
MDG-a_2_n500_m50	7605.52	2.14	0.00734
MDG-a_3_n500_m50	7640.15	1.54	0.004435
MDG-a_4_n500_m50	7506.45	3.39	0.002631
MDG-a_5_n500_m50	7595.3	2.06	0.007672
MDG-a_6_n500_m50	7513.32	3.35	0.002595
MDG-a_7_n500_m50	7531.44	3.09	0.00346
MDG-a_8_n500_m50	7609.81	1.82	0.007511
MDG-a_9_n500_m50	7654.97	1.48	0.006011
MDG-a_10_n500_m50	7707.51	0.94	0.012247
MDG-b_21_n2000_m200	11155738.985299	1.28	0.186285
MDG-b_22_n2000_m200	11164122.435074	1.09	0.184283
MDG-b_23_n2000_m200	11177722.312113	1.08	0.180047
MDG-b_24_n2000_m200	11124330.872795	1.48	0.175191
MDG-b_25_n2000_m200	11181796.14402	1.01	0.175225
MDG-b_26_n2000_m200	11157826.838857	1.19	0.176321
MDG-b_27_n2000_m200	11165835.734422	1.24	0.177808
MDG-b_28_n2000_m200	11119034.565327	1.43	0.174686
MDG-b_29_n2000_m200	11163768.720372	1.18	0.175868
MDG-b_30_n2000_m200	11148372.956175	1.31	0.174758
MDG-c_1_n3000_m300	24637456	0.99	0.418843
MDG-c_2_n3000_m300	24595823	1.24	0.416247
MDG-c_8_n3000_m400	43042008	0.91	0.515977
MDG-c_9_n3000_m400	43044732	0.91	0.51509
MDG-c_10_n3000_m400	43031426	1.02	0.514451
MDG-c_13_n3000_m500	66556889	0.68	0.602169
MDG-c_14_n3000_m500	66646003	0.5	0.598817
MDG-c_15_n3000_m500	66693908	0.45	0.605126
MDG-c_19_n3000_m600	95013094	0.65	0.672743
MDG-c_20_n3000_m600	94941193	0.73	0.664133

Tabla 4: Evaluación de las soluciones y estadísticos *Desv* y *Tiempo* obtenidos por el algoritmo de Enfriamiento Simulado con Cauchy modificado en cada caso de estudio.

Media de los estadísticos:

Desv	Tiempo (s)
1.41	0.25

Enfriamiento Simulado (Proporcional)

Caso	Coste obtenido	Desv	Tiempo (s)
MDG-a_1_n500_m50	7822.51	0.14	0.017657
MDG-a_2_n500_m50	7622.47	1.92	0.017737
MDG-a_3_n500_m50	7687.45	0.93	0.018346
MDG-a_4_n500_m50	7721.95	0.62	0.018346
MDG-a_5_n500_m50	7691.9	0.82	0.017978
MDG-a_6_n500_m50	7725.73	0.62	0.017659
MDG-a_7_n500_m50	7666.35	1.36	0.018872
MDG-a_8_n500_m50	7654.96	1.24	0.018646
MDG-a_9_n500_m50	7700.72	0.89	0.018478
MDG-a_10_n500_m50	7747.54	0.42	0.01974
MDG-b_21_n2000_m200	10162614.655392	10.06	0.209395
MDG-b_22_n2000_m200	10173535.392627	9.86	0.210087
MDG-b_23_n2000_m200	10155768.757838	10.13	0.210103
MDG-b_24_n2000_m200	10153786.637714	10.07	0.209786
MDG-b_25_n2000_m200	10205945.840977	9.65	0.209136
MDG-b_26_n2000_m200	10186840.609219	9.79	0.209538
MDG-b_27_n2000_m200	10178369.828872	9.97	0.210577
MDG-b_28_n2000_m200	10165757.250428	9.88	0.209549
MDG-b_29_n2000_m200	10166804.326253	10.01	0.20924
MDG-b_30_n2000_m200	10167455.620431	9.99	0.210194
MDG-c_1_n3000_m300	22710993	8.73	0.367478
MDG-c_2_n3000_m300	22674638	8.96	0.365672
MDG-c_8_n3000_m400	40198770	7.46	0.44825
MDG-c_9_n3000_m400	40192167	7.47	0.448765
MDG-c_10_n3000_m400	40208739	7.52	0.459702
MDG-c_13_n3000_m500	62739982	6.38	0.524618
MDG-c_14_n3000_m500	62722779	6.36	0.521154
MDG-c_15_n3000_m500	62827242	6.22	0.519484
MDG-c_19_n3000_m600	90250861	5.63	0.576816
MDG-c_20_n3000_m600	90300816	5.59	0.576425

Tabla 5: Evaluación de las soluciones y estadísticos *Desv* y *Tiempo* obtenidos por el algoritmo de Enfriamiento Simulado Proporcional en cada caso de estudio.

Media de los estadísticos:

Desv	Tiempo (s)
5.96	0.24

Búsqueda Multiarranque Básica

Caso	Coste obtenido	Desv	Tiempo (s)
MDG-a_1_n500_m50	7728.33	1.35	0.012555
MDG-a_2_n500_m50	7660.38	1.43	0.012736
MDG-a_3_n500_m50	7685.73	0.95	0.012163
MDG-a_4_n500_m50	7670.73	1.28	0.013113
MDG-a_5_n500_m50	7697.13	0.75	0.01369
MDG-a_6_n500_m50	7672.51	1.3	0.012915
MDG-a_7_n500_m50	7671.47	1.29	0.012711
MDG-a_8_n500_m50	7638.95	1.44	0.012538
MDG-a_9_n500_m50	7671.42	1.27	0.013783
MDG-a_10_n500_m50	7691.73	1.14	0.015372
MDG-b_21_n2000_m200	11191731.408656	0.96	0.548181
MDG-b_22_n2000_m200	11173811.056111	1	0.545127
MDG-b_23_n2000_m200	11188292.385945	0.99	0.536318
MDG-b_24_n2000_m200	11171816.301443	1.05	0.547428
MDG-b_25_n2000_m200	11187435.832728	0.96	0.545627
MDG-b_26_n2000_m200	11183285.709461	0.97	0.539269
MDG-b_27_n2000_m200	11187635.60774	1.04	0.527482
MDG-b_28_n2000_m200	11159849.337533	1.06	0.521256
MDG-b_29_n2000_m200	11165563.257994	1.17	0.519606
MDG-b_30_n2000_m200	11175655.345143	1.07	0.526765
MDG-c_1_n3000_m300	24669394	0.86	2.025703
MDG-c_2_n3000_m300	24640413	1.06	2.012946
MDG-c_8_n3000_m400	43103744	0.77	4.708329
MDG-c_9_n3000_m400	43115518	0.74	4.693528
MDG-c_10_n3000_m400	43106298	0.85	4.674107
MDG-c_13_n3000_m500	66572923	0.66	8.603123
MDG-c_14_n3000_m500	66605573	0.56	8.606731
MDG-c_15_n3000_m500	66616585	0.56	8.536043
MDG-c_19_n3000_m600	95117057	0.54	13.523184
MDG-c_20_n3000_m600	95136282	0.53	13.541991

Tabla 6: Evaluación de las soluciones y estadísticos *Desv* y *Tiempo* obtenidos por el algoritmo de Búsqueda Multiarranque Básica en cada caso de estudio.

Media de los estadísticos:

Desv	Tiempo (s)
0.99	2.55

Búsqueda Local Reiterada

Caso	Coste obtenido	Desv	Tiempo (s)
MDG-a_1_n500_m50	7709.93	1.58	0.005458
MDG-a_2_n500_m50	7724.48	0.61	0.005492
MDG-a_3_n500_m50	7700.95	0.75	0.005432
MDG-a_4_n500_m50	7698.72	0.92	0.004801
MDG-a_5_n500_m50	7675.08	1.03	0.005607
MDG-a_6_n500_m50	7724.5	0.63	0.00516
MDG-a_7_n500_m50	7684.35	1.12	0.005209
MDG-a_8_n500_m50	7636.8	1.47	0.004604
MDG-a_9_n500_m50	7588.69	2.33	0.005811
MDG-a_10_n500_m50	7639.14	1.81	0.005155
MDG-b_21_n2000_m200	11246174.943997	0.48	0.279171
MDG-b_22_n2000_m200	11234217.903533	0.47	0.283377
MDG-b_23_n2000_m200	11265404.407986	0.31	0.282715
MDG-b_24_n2000_m200	11235595.316972	0.49	0.278637
MDG-b_25_n2000_m200	11238488.875162	0.51	0.264429
MDG-b_26_n2000_m200	11222485.977794	0.62	0.266735
MDG-b_27_n2000_m200	11270651.986493	0.31	0.271429
MDG-b_28_n2000_m200	11206855.439177	0.65	0.267736
MDG-b_29_n2000_m200	11232129.73934	0.58	0.28129
MDG-b_30_n2000_m200	11221257.950393	0.67	0.268129
MDG-c_1_n3000_m300	24799202	0.34	0.891335
MDG-c_2_n3000_m300	24768855	0.55	0.873298
MDG-c_8_n3000_m400	43289507	0.34	1.848681
MDG-c_9_n3000_m400	43310665	0.29	1.906863
MDG-c_10_n3000_m400	43295479	0.42	1.905886
MDG-c_13_n3000_m500	66746057	0.4	3.34356
MDG-c_14_n3000_m500	66829831	0.22	3.357704
MDG-c_15_n3000_m500	66888838	0.16	3.323805
MDG-c_19_n3000_m600	95338561	0.31	5.247
MDG-c_20_n3000_m600	95405963	0.25	5.039759

Tabla 7: Evaluación de las soluciones y estadísticos *Desv* y *Tiempo* obtenidos por el algoritmo de Búsqueda Local Reiterada en cada caso de estudio.

Media de los estadísticos:

Desv	Tiempo (s)
0.69	1.02

Híbrido ILS-ES (Cauchy Modificado)

Caso	Coste obtenido	Desv	Tiempo (s)
MDG-a_1_n500_m50	7668.88	2.11	0.018958
MDG-a_2_n500_m50	7570.89	2.58	0.019275
MDG-a_3_n500_m50	7629.52	1.67	0.019115
MDG-a_4_n500_m50	7610.35	2.06	0.019151
MDG-a_5_n500_m50	7558.51	2.54	0.019174
MDG-a_6_n500_m50	7611.88	2.08	0.01904
MDG-a_7_n500_m50	7595.01	2.27	0.018986
MDG-a_8_n500_m50	7608.75	1.83	0.019084
MDG-a_9_n500_m50	7637.98	1.7	0.019134
MDG-a_10_n500_m50	7624.97	2	0.02031
MDG-b_21_n2000_m200	11102518.768872	1.75	0.417906
MDG-b_22_n2000_m200	11097704.226095	1.68	0.434521
MDG-b_23_n2000_m200	11107942.10879	1.7	0.422667
MDG-b_24_n2000_m200	11095329.156269	1.73	0.418913
MDG-b_25_n2000_m200	11106612.269199	1.68	0.407629
MDG-b_26_n2000_m200	11115693.739341	1.56	0.437994
MDG-b_27_n2000_m200	11106412.843644	1.76	0.38352
MDG-b_28_n2000_m200	11099474.468795	1.6	0.377382
MDG-b_29_n2000_m200	11090079.958744	1.83	0.381952
MDG-b_30_n2000_m200	11108647.770283	1.66	0.381864
MDG-c_1_n3000_m300	24518332	1.47	1.087397
MDG-c_2_n3000_m300	24539459	1.47	1.088532
MDG-c_8_n3000_m400	42966259	1.08	1.346259
MDG-c_9_n3000_m400	42961477	1.1	1.343878
MDG-c_10_n3000_m400	42931845	1.25	1.340798
MDG-c_13_n3000_m500	66388188	0.93	1.545965
MDG-c_14_n3000_m500	66365061	0.92	1.549764
MDG-c_15_n3000_m500	66443388	0.82	1.548566
MDG-c_19_n3000_m600	94850218	0.82	1.725036
MDG-c_20_n3000_m600	94801190	0.88	1.74827

Tabla 8: Evaluación de las soluciones y estadísticos *Desv* y *Tiempo* obtenidos por el algoritmo que combina Búsqueda Local Reiterada con Enfriamiento Simulado (con Cauchy Modificado) en cada caso de estudio.

Media de los estadísticos:

Desv	Tiempo (s)
1.62	0.62

Híbrido ILS-ES (Proporcional)

Caso	Coste obtenido	Desv	Tiempo (s)
MDG-a_1_n500_m50	7560.83	3.48	0.01891
MDG-a_2_n500_m50	7563.4	2.68	0.018826
MDG-a_3_n500_m50	7460.16	3.86	0.019232
MDG-a_4_n500_m50	7451.87	4.1	0.01887
MDG-a_5_n500_m50	7436.75	4.11	0.019018
MDG-a_6_n500_m50	7518.25	3.29	0.018643
MDG-a_7_n500_m50	7502.04	3.47	0.018787
MDG-a_8_n500_m50	7495.04	3.3	0.018996
MDG-a_9_n500_m50	7554.54	2.77	0.018889
MDG-a_10_n500_m50	7462.38	4.09	0.0205
MDG-b_21_n2000_m200	10057595.617523	10.99	0.192389
MDG-b_22_n2000_m200	10063542.205813	10.84	0.192122
MDG-b_23_n2000_m200	9993237.92958	11.56	0.192491
MDG-b_24_n2000_m200	10021100.549492	11.25	0.192892
MDG-b_25_n2000_m200	10025313.148476	11.25	0.192684
MDG-b_26_n2000_m200	10051312.29967	10.99	0.193354
MDG-b_27_n2000_m200	10051140.804994	11.1	0.192243
MDG-b_28_n2000_m200	10025336.809131	11.12	0.193467
MDG-b_29_n2000_m200	10047516.382319	11.06	0.192314
MDG-b_30_n2000_m200	9971925.528861	11.72	0.19205
MDG-c_1_n3000_m300	22502085	9.57	0.405454
MDG-c_2_n3000_m300	22535949	9.51	0.407438
MDG-c_8_n3000_m400	40024366	7.86	0.499524
MDG-c_9_n3000_m400	40134952	7.6	0.497438
MDG-c_10_n3000_m400	40082879	7.81	0.5004
MDG-c_13_n3000_m500	62482702	6.76	0.576292
MDG-c_14_n3000_m500	62517590	6.66	0.571553
MDG-c_15_n3000_m500	62559375	6.62	0.570204
MDG-c_19_n3000_m600	90059376	5.83	0.63464
MDG-c_20_n3000_m600	89993465	5.91	0.633904

Tabla 9: Evaluación de las soluciones y estadísticos *Desv* y *Tiempo* obtenidos por el algoritmo que combina Búsqueda Local Reiterada con Enfriamiento Simulado (Proporcional) en cada caso de estudio.

Media de los estadísticos:

Desv	Tiempo (s)
7.37	0.25

Comparamos los estadísticos medios obtenidos estos algoritmos entre sí y con los obtenidos por los algoritmos de búsqueda local (con primer mejor) y greedy de la primera práctica.

Algoritmo	Desv	Tiempo (s)
Greedy	1.63	1.19
BL	1.3	0.69
ES-CM	1.41	0.25
ES-prop	5.96	0.24
BMB	0.99	2.55
ILS	0.69	1.02
ILS-ES-CM	1.62	0.62
ILS-ES-prop	7.37	0.25

Tabla 10: Comparativa de los estadísticos medios obtenidos por los distintos algoritmos.

6.2. Análisis de resultados

Tiempos

Lo primero que debemos destacar es que los tiempos de estos algoritmos son muy inferiores a los obtenidos por los algoritmos poblacionales de la práctica anterior (cuyo tiempo medio oscilaba entre 40 y 90 segundos), esto se debe principalmente a que todos los algoritmos basados en trayectorias utilizan la factorización de la función de fitness. El algoritmo que más tiempo consume es BMB, supongo que por tener que generar varias soluciones aleatorias. Sin embargo, su tiempo de cómputo también es prácticamente despreciable en comparación con los algoritmos poblacionales.

Enfriamiento Proporcional vs Cauchy Modificado

El esquema de enfriamiento de Cauchy Modificado enfría muy rápido al principio. En los ejemplos del grupo MDG-a, la temperatura inicial ronda los 1400 ó 1500, y tras un enfriamiento es cercana a 0.2, siendo los posteriores enfriamientos más paulatinos. Es por esto que sospechamos que el Enfriamiento Proporcional, más paulatino, iba a obtener mejores resultados. Ajustamos un coeficiente de enfriamiento $\alpha = 0.9$ a ojo, observando un par de ejecuciones del grupo MDG-a con otras semillas distintas, lo cual ha resultado ser un fracaso.

Como podemos observar en las Tablas 4 y 5, el enfriamiento proporcional obtiene desviaciones mucho más bajas en los ejemplos del grupo MDG-a, pero su desempeño es desastroso en los ejemplos de los grupos MDG-b y MDG-c.

Para un grupo de ejemplos concretos, podemos encontrar un coeficiente de modo que el enfriamiento proporcional supere a Cauchy modificado, al menos en los grupos MDG-a y MDG-c, donde todos los ejemplos tienen los mismos parámetros. Sin embargo, la gran ventaja del enfriamiento proporcional es que adapta sus parámetros al número de iteraciones esperado (en función del número de evaluaciones y de la dimensionalidad del problema), mientras que un valor fijo para el enfriamiento proporcional no produce resultados satisfactorios para los posibles distintos tamaños del problema.

En la Tabla 5 observamos que los ejemplos del grupo MDG-b son los peores para el enfriamiento proporcional. En los ejemplos del grupo MDG-c, la desviación (sin dejar de ser demasiado alta) va decreciendo cuando mantenemos el parámetro n y vamos incrementando el m, lo que parece indicar que su desempeño es peor cuanto menor sea (en proporción) el subconjunto a seleccionar, al menos para el parámetro $\alpha = 0.9$.

En la hibridación de ES con ILS, el número de iteraciones es menor en cada ejecución de ES, por lo que Cauchy modificado se adapta enfriando aún más rápido mientras que tenemos dificultades para ajustar el parámetro $\alpha = 0.5$, con el que ocurre lo mismo que en el caso anterior, esta vez incluso con resultados peores en los ejemplos del grupo MDG-a (Tablas 8 y 9).

En tiempo no hay mucha diferencia en la ejecución, pero sí que la hay en la inicialización de los parámetros (el esquema proporcional no requiere inicializar M, T_f y $beta$), esto se aprecia al combinarlo con ILS, ya que la inicialización se realiza 10 veces y el esquema proporcional saca una ventaja de tiempo, que por supuesto no compensa su alta desviación.

Este experimento nos ha servido para apreciar la facilidad de adaptación del esquema de Cauchy modificado a la hora de enfriar. A partir de ahora, siempre consideraremos este esquema. Ignoraremos los algoritmos ES-prop y ILS-ES-prop por el resto del análisis.

Búsqueda Local vs Enfriamiento Simulado

Primero los comparamos en sus versiones simples: Tabla 3 frente a Tabla 4.

El algoritmo de Enfriamiento Simulado es bastante más rápido que el de Búsqueda Local, esto se debe principalmente a que ES busca vecinos aleatorios mientras LS tiene que calcular el menor contribuyente para cada salto. Esto hace que los saltos de LS produzcan una mayor mejora en la fitness, por lo que sus 100000 iteraciones se invierten en mejorar la solución en mayor medida que las de ES.

El desempeño de ES es ligeramente peor, pero como acabamos de explicar las iteraciones de LS son más costosas, de modo que quizá deberíamos haber permitido a ES realizar más iteraciones para que esta comparación fuese más justa.

La cosa cambia cuando los consideramos en combinación con ILS: Tablas 7 y 8.

A pesar de que ES sigue siendo ligeramente más rápido, esta diferencia se nota menos en este caso. Probablemente es debido a que la inicialización de ES es más lenta, y aquí se tiene que inicializar 10 veces en lugar de 1.

Además, aquí si hay una gran mejora de la Búsqueda Local. El ES permite que la solución empeore en sus primeras etapas, y el número de iteraciones en cada ejecución es demasiado bajo. Aunque el esquema de Cauchy Modificado permite que se adapte para enfriar más rápido ante un menor número de iteraciones, no tiene suficientes iteraciones para que la fase de explotación (cuando la temperatura es muy baja) dé sus frutos.

Aun así, ocurre el mismo problema que comentamos antes, puede que esta comparación no sea justa del todo, ya que una iteración de LS es bastante más costosa por tener que calcular el menor contribuyente. Sin embargo, en esta ocación la diferencia de tiempo no compensa en absoluto la diferencia de desviación.

Trayectorias Simples vs Múltiples

Como hemos comentado antes, todos los algoritmos estudiados en esta práctica son relativamente rápidos. Sin embargo (ignorando el Greedy, que no es de trayectoria), los algoritmos basados en trayectorias múltiples son algo más lentos que los basados en trayectorias múltiples: ILS-ES-CM es más lento que ES-CM y ILS y BMB son más lentos que BL. Esto se debe a que necesitan inicializar la búsqueda 10 veces en lugar de sólo 1, y generar una solución aleatoria lleva tiempo.

En cuanto a fitness, los algoritmos de trayectoria múltiple (exceptuando a ILS-ES, hemos comentado anteriormente la razón de que estos algorimtos no tengan buena sinergia) superan no solo a los de trayectorias simples sino también a todos los poblacionales que implementamos en la práctica anterior (el mejor fue un memético que consiguió 1.09 de desviación). Por tanto, podemos concluir que merece más la pena realizar varios intentos de búsqueda local que invertir todas las evaluaciones en mejorar una solución lo máximo posible. Es mejor aumentar la posibilidad de caer "cerca" de un máximo local muy alto que alcanzar el máximo y que no sea tan alto.

ILS vs BMB

Como acabamos de comentar, parece provechoso no invertir todas las evaluaciones en mejorar una solución lo máximo posible. Sin embargo, en lugar de realizar intentos aleatorios independientes como BMB, ha resultado bastante mejor partir desde una mutación (bastante fuerte) de la mejor solución encontrada hasta el momento. Es decir, de una solución (medianamente lejana) de su entorno. Esto es lo que hace ILS y supera con cierto margen al resto de algoritmos que hemos estudiado.

En cuanto a tiempo, ILS es mucho más rápido que BMB. Esto se debe a los tiempos que necesitan para inicializar cada búsqueda. Es más costoso generar una solución aleatoria nueva que mutar una solución existente.