# METAHEURÍSTICAS

*PRÁCTICA 1.A: TÉCNICAS DE BÚSQUEDA LOCAL Y ALGORITMOS GREEDY PARA EL PROBLEMA DE LA MÁXIMA DIVERSIDAD.*

***Curso 2019-2020***

*Tercer curso del Grado en Ingeniería Informática*

Iván Cortón da Silva 3ºCSI
15426623-V
ivancorton99@correo.ugr.es
Grupo MH2: jueves de 17.30h a 19.30h

# ÍNDICE:

# PRÁCTICA 1 – TÉCNICAS DE BÚSQUEDA LOCAL Y ALGORITMOS GREEDY PARA EL PROBLEMA DE LA MÁXIMA DIVERSIDAD.

Con el fin de llevar a cabo el correcto desarrollo de la primera práctica de la asignatura, se nos pide el estudio del funcionamiento de *Técnicas de Búsqueda Local y de los Algoritmos Greedy,* en concreto, en la resolución del problema de la máxima diversidad (MDP) descrito en el Seminario 2.

Para ello, a continuación, expondré las adaptaciones de dichas técnicas al problema elegido, en el que finalmente realizaremos una comparación de los resultados obtenidos con las estimaciones convenientes para el valor de los óptimos de una serie de casos del problema.

El problema de la máxima diversidad, está caracterizado por ser un problema de optimización combinatoria que consiste en seleccionar un subconjunto M de m elementos (|M| = m) de un conjunto inicial N de n elementos (con n > m) de forma que se podría formular como sigue:

$$\text{Maximizar} \quad z_{MS}(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j$$

$$\text{Sujeto a} \quad \sum_{i=1}^{n} x_i = m$$

$$x_i = \{0, 1\}, \quad i = 1, \ldots, n.$$

donde:

- X es un vector solución al problema binario que indica los m elementos seleccionados.
- $d_{ij}$ es la distancia existente entre los elementos i-ésimos y j-ésimos.

Por otro lado, respecto a los casos a poner a prueba dentro de cada tipo de enfoque del problema (Greedy y Búsqueda Local), se utilizarán un total de **30 casos** seleccionados de varios de los conjuntos de instancias disponibles en la MDPLIB (http://www.optsicom.es/mdp/) y en la plataforma Prado.

Entre estos **30 casos**, encontramos 10 pertenecientes al grupo **GKD** con **distancias Euclideas** relativamente pequeñas, con n=500 y m=50 (GKD-c_11_n500_m50 a GKDc_20_n500_m50). Un segundo grupo **MDG** caracterizado por utilizar **distancias reales** en [0,1000], con n=500 y m=50 (MDG-b_1_n500_m50 a MDG-b_10_n500_m50); y por último otras 10 del grupo **MDG** con **distancias de números enteros** entre 0 y 10 con n=2000 y m=200 (MDG-a_31_n2000_m200 a MDGa_40_n2000_m200).

## ENFOQUE GREEDY

Primeramente, y por simplicidad, realizaremos el análisis del procedimiento Greedy llevado a cabo para obtener resolver el problema de la máxima diversidad.

Este algoritmo se basa en la heurística de ir seleccionando los **elementos más lejanos a los seleccionados** previamente, es decir, nuestro primer elemento será aquel cuya distancia a todos los demás no seleccionados sea máxima. A continuación, en los m-1 pasos siguientes iremos escogiendo el elemento más lejano a los elementos **seleccionados** hasta el momento.

Por tanto, podemos determinar que una buena fórmula heurística para resolver el problema consiste en, ir añadiendo secuencialmente el elemento que no haya sido seleccionado que más diversidad aporte **únicamente** respecto a los ya seleccionados.

```
Inicio
 1. Sel = Ø
 2. nodoObjetivo = Calc(noSel)
 3. Sel = Sel ∪ {nodoObjetivo}
 4. NoSel – {nodoObjetivo}
     while (|Sel| < m)
     5. Añadir i* /  {d(s_i*,s_j) = max{d(s_i,s_j)}, s_j ∈ Sel,
     s_i ∉ Sel}
     //Empleamos un método de búsqueda sobre Sel para
     //evitar adición de nodos ya añadidos.
     6. NoSel = NoSel – {s_i*}
     end while
 Fin
```

Pseudocódigo del algoritmo Greedy

**donde:**

- "Calc" es una función encargada de elegir el primer nodo tras realizar el cálculo y comparación de las distancias desde cada nodo i a cada otro nodo j / i,j ∈ [0,n[.
- El método de búsqueda sobre Sel para evitar adición de nodos ya incluidos en la solución, consiste en una búsqueda a través del vector solución del índice a identificar.

```
Inicio
 1. valorActual = 0
 2. valorAnterior = 0
 3. candidato = 0
 4. Desde i=0 hasta n hacer
      4. Desde j=0 hasta n hacer
      5. valorActual = valorActual + d_{i,j}
         6. if valorActual > valorAnterior
               7. ValorActual = valorAnterior
               8. Candidato = i
            endif
         Siguiente
    Siguiente
 Fin
```

Pseudocódigo de la función Calc

Por otro lado, en referencia a la **representación de la solución**, utilizaremos un vector de enteros que indicarán los índices de los nodos que mayor aporte hagan a la solución encontrada hasta el momento, calculando la distancia de cada nodo de la solución a cada uno de los demás existentes en este vector solución.

En cuanto al análisis de datos obtenidos en comparación con la tabla de resultados aportados por el profesor de la asignatura en la plataforma Prado, podemos comprobar notorias diferencias que analizaremos posteriormente:

| Algoritmo Greedy | | | |
|---|---|---|---|
| **Caso** | **Coste obtenido** | **Desv** | **Tiempo** |
| GKD-c_11_n500_m50 | 19583,862 | 0,02 | 4,90 |
| GKD-c_12_n500_m50 | 19349,2 | 0,06 | 4,90 |
| GKD-c_13_n500_m50 | 19348,8 | 0,09 | 4,90 |
| GKD-c_14_n500_m50 | 19458,37001 | 0,00 | 4,90 |
| GKD-c_15_n500_m50 | 19420,26198 | 0,01 | 4,90 |
| GKD-c_16_n500_m50 | 19677,32826 | 0,01 | 4,90 |
| GKD-c_17_n500_m50 | 19331,388 | 0,00 | 4,90 |
| GKD-c_18_n500_m50 | 19461,39453 | 0,00 | 4,90 |
| GKD-c_19_n500_m50 | 19472,816 | 0,02 | 4,90 |
| GKD-c_20_n500_m50 | 19604,8435 | 0,00 | 4,90 |
| MDG-b_1_n500_m50 | 754034,9599 | 3,08 | 5,10 |
| MDG-b_2_n500_m50 | 759842,599 | 2,58 | 5,10 |
| MDG-b_3_n500_m50 | 757355,42 | 2,50 | 5,10 |
| MDG-b_4_n500_m50 | 763215,8799 | 1,57 | 5,10 |
| MDG-b_5_n500_m50 | 753856,6499 | 2,80 | 5,10 |
| MDG-b_6_n500_m50 | 756625,31 | 2,39 | 5,10 |
| MDG-b_7_n500_m50 | 757362,13 | 2,56 | 5,10 |
| MDG-b_8_n500_m50 | 759993,629 | 2,46 | 5,10 |
| MDG-b_9_n500_m50 | 754165,72 | 2,66 | 5,10 |
| MDG-b_10_n500_m50 | 753517,389 | 2,77 | 5,10 |
| MDG-a_31_n2000_m200 | 112505 | 1,43 | 600,00 |
| MDG-a_32_n2000_m200 | 112425 | 1,46 | 600,00 |
| MDG-a_33_n2000_m200 | 112375 | 1,53 | 600,00 |
| MDG-a_34_n2000_m200 | 112262 | 1,70 | 600,00 |
| MDG-a_35_n2000_m200 | 112558 | 1,42 | 600,00 |
| MDG-a_36_n2000_m200 | 112355 | 1,66 | 600,00 |
| MDG-a_37_n2000_m200 | 112453 | 1,54 | 600,00 |
| MDG-a_38_n2000_m200 | 112495 | 1,65 | 600,00 |
| MDG-a_39_n2000_m200 | 112283 | 1,68 | 600,00 |
| MDG-a_40_n2000_m200 | 112709 | 1,30 | 600,00 |

**Media Desv:** **1,37**

**Media Tiempo:** **203,33**

A la hora de analizar los tiempos, desviaciones y costes obtenidos durante la implementación del enfoque Greedy, tuve que atender a una serie de características que al comienzo no tuve muy en cuenta y que me llevo a cambiar el enfoque de la resolución del problema.

En principio, planteé el problema, como un problema de optimización que una vez encontraba un nodo que hiciera la distancia máxima entre todos los demás nodos (sin seleccionar), únicamente tendríamos que ir escogiendo los siguientes nodos que estuvieran más alejados de todos los demás nodos y **no estuvieran seleccionados aún**. Este enfoque hizo que perdiera una gran cantidad de tiempo hasta caer en que los costes obtenidos no eran parecidos ni por asomo a los aportados por el profesor.

Posteriormente, tras varios análisis del problema y discusiones con otros compañeros, comprendí que el problema trataba de encontrar **aquellos más alejados de todos los demás** (en el caso del primer elegido), seguidos de los **más alejados de todos los ya seleccionados (operador de selección del vecindario)**, lo cual me hizo entrar un poco en razón y comenzar a obtener resultados válidos, o al menos próximos a los del profesor.

Varios son los aspectos que creo que influyeron en la obtención de mis resultados, provocando así una desviación >1 y un tiempo tan abrumador:

En primer lugar, y referente a la desviación, me encontré que los datos que obtuve en problemas de tamaño n=500 y m=50 de tipo MDG (es decir, con tamaños en [0,1000]) estaban **bastantes alejados** de los valores de **costes óptimos obtenidos** y creo que quizás tendría que ver con, o bien el criterio para elegir el primer nodo, provocando así un primer nodo de distancia máxima erróneo; o bien con el tamaño del problema junto con el tipo de dato empleado, que produzca algún tipo de ralentización o comportamiento anómalo dentro del código; o bien por el uso de una máquina virtual a través de la que estuve realizando las primeras ejecuciones de prueba del programa para poder comprobar si verdaderamente el fallo estaba en ella.

Obtener estas conclusiones se debe a la diferencia encontrada a simple vista entre los problemas de tipo GKD y MDG, ya que en los problemas MDG con n=2000 no encontramos una desviación tan abismal como en la MDG con n=500, al igual que también encontramos diferencias entre los tiempos obtenidos en los GKD y MDG con n=500, y aquellos obtenidos con n=2000; que creo que puede estar relacionado con tratar el tipo de dato entero mediante variables de tipo “auto”, o simplemente debido a que el tamaño del problema crece factorialmente.

Luego, traté de trasladar toda la información al Sistema Operativo de mi equipo, sin el uso de máquinas virtuales ni ningún otro tipo de software capaz de incrementar el consumo de nuestra memoria y CPU, instalando y usando el entorno de programación Visual Studio Code. Tras realizar la implementación y ejecución del código, no pude ver una diferencia lo suficientemente significativa como para poder decir que fuera culpa de la máquina virtual, aunque sí una mejoría en los tiempos obtenidos de en torno unos 3 o 4 segundos.

Finalmente, y tras la implementación del enfoque de Búsqueda Local por el primer mejor junto con varias discusiones con algunos compañeros, pude comprobar que el causante de obtener tiempos tan elevados probablemente pudiera ser el tipo de estructura de datos usada (vector), que frente a la estructura de datos tipo lista, queda bastante en desventaja en cuanto al tiempo de acceso a los elementos de dicha estructura, mediante el uso de iteradores. Cabe destacar, además, la velocidad aportada por la factorización de los cálculos de los aportes de cada candidato que trataré de explicar a continuación.

## ENFOQUE BÚSQUEDA LOCAL PRIMER MEJOR

Una vez analizado y aclarado el comportamiento del *enfoque Greedy,* trataré de explicar el segundo enfoque encomendado a esta primera práctica.

En primer lugar, me gustaría concretar el formato de representación de la solución, que análogamente al enfoque anterior, reuniremos en una lista de enteros aquellos vecinos que realicen un mayor aporte de diversidad respecto a los demás nodos dentro de la solución, quedando de la forma:

***Sel = {$x_0$ , $x_1$ ,…, $x_m$}*** que se encargará de almacenar los **m** elementos de **n** que forman la lista solución.

Al igual que en el enfoque anterior, una solución será factible siempre y cuando; no presente elementos repetidos, contenga exactamente m elementos, y el orden de los elementos no importe.

Tras aclarar el formato de representación de soluciones, analizaremos la manera de generar dichas soluciones de la forma más clara y concisa posible que en un primer momento, también tuve bastantes problemas para poder entender completamente en qué consistía este enfoque.

**Una primera aproximación** que implementé para tratar de resolver este problema consistió en, elegir primeramente una solución generada aleatoriamente (mediante el uso del fichero aportado por los profesores en la plataforma Prado), y posteriormente ir comprobando el aporte individual de cada uno de los elementos no incluidos en nuestra lista solución, quedándonos así con aquel elemento que garantizara un aporte a la solución mayor que el mínimo aporte encontrado en nuestra solución. Este método de obtención de soluciones, en principio parecía encontrar soluciones bastante factibles, incluso mejores que las soluciones obtenidas por el *enfoque Greedy,* aunque tras varios análisis tanto del código como de la técnica exigida para su resolución, pude comprobar que realmente estaba realizando una especie de implementación **del *enfoque Greedy*,** siendo la única diferencia que, en un principio, la lista solución se encontraría llena de elementos aleatorios.

Más tarde, tras preguntar a varios compañeros en busca de aclaraciones de la explicación del enfoque, pude denotar que mi fallo realmente fue entender este enfoque como que podíamos realizarlo con **“dos variantes”**:

- **Búsqueda local mediante la técnica del primer mejor exclusivamente**, sin atender a la exploración inteligente del vecindario.
- **Búsqueda local mediante la técnica de generación inteligente del vecindario únicamente**, sin atender al primer mejor encontrado, sino que buscando siempre aquel vecino con el mejor aporte a la solución encontrada hasta el momento.

Al igual que en el enfoque anterior, este desconocimiento acerca de las técnicas exigidas para abordar el Problema de la Máxima Diversidad me hizo perder una gran cantidad de tiempo, ya que traté de implementar ambas “variantes”, aunque pude comprobar que, igualmente se podían obtener resultados con costes mejores en comparación con el *enfoque Greedy.*

Finalmente, y tras la ayuda de algún compañero, pude comprender que verdaderamente, se nos pedía implementar aquello que creía como "variantes", conjuntamente para poder optimizar tanto costes como tiempos obteniendo soluciones mejores en tiempos bastante más razonables que los obtenidos por el *Greedy*, añadiendo además cambios que serían muy significativos; como el hecho del cambio de estructura de datos de **std::vector** a **std::list,** accediendo a través de iteradores y utilizando varias listas diferenciando tanto índices como aportes de cada uno de los elementos de la solución, o el hecho de incluir la factorización de los aportes, que reducían significativamente los tiempos de ejecución obtenidos, incluso incluyendo un método que fuera capaz de ordenar nuestra lista solución para posteriormente desordenarla aleatoriamente evitando la comprobación secuencial de elementos en función de su aporte.

A continuación, trataré de representar en forma de pseudocódigo, el procedimiento realizado para la aplicación del algoritmo de Búsqueda Local por el Primer Mejor:

```
Inicio
 1. Sel = generarSolucionAleatoria
 2. noSel = noSel – {Sel}
 3. while (hayMejora y cambio < limite)
       4. aportes = factorizacionAportes
       5. ordenación de lista de seleccionados y aportes
       6. while (vecindarioNoExplorado y noHayMejora)
             7. barajar noSel
             8. Desde i = inicioNoSel hasta finalNoSel hacer
                  9. quitar candidato a salir de Sel y buscar
                  mejor candidato en noSel.
             Siguiente
             10. if noHayMejora
                  11. comprobar siguiente elemento en Sel
             endif
           endwhile
           12. cambio++
    endwhile
 Fin
```

Pseudocódigo del algoritmo Búsqueda Local

```
Inicio
 1. while (vecindarioNoExplorado y noHayMejora)
    2. barajar noSel
    3. Desde i = inicioNoSel hasta finalNoSel hacer
       4. quitar candidato a salir de Sel y buscar mejor
       candidato en noSel.
    Siguiente
    5. if noHayMejora
        6. comprobar siguiente elemento en Sel
      endif
  endwhile
Fin
```

Pseudocódigo de la exploración del entorno

```
Inicio
      1. Transformar noSel en vector
      2. Aplicación método std::random_shuffle
      3. Sobreescribir noSel con datos en vector
Fin
```

Pseudocódigo de la generación del vecindario (método barajar).

```
Inicio
      1. Desde it = inicioSel hasta finalSel
      2. j = i   //Usado para sumar en el índice
                 correspondiente i y en el índice j que va
                 avanzando.
         3. Desde jt = it hasta finalSel
            4. aporte en el índice i += d_{it,jt}
            5. aporte en el índice j += d_{it,jt}
         Siguiente
      Siguiente
Fin
```

Pseudocódigo de la factorización de los cálculos.

En primer lugar, nos encontramos con el pseudocódigo general seguido para poder realizar la implementación correctamente de la metaheurística.

Al comienzo, rellenamos nuestra lista de seleccionados con un conjunto aleatorio de elementos (pertenecientes a los candidatos posibles a formar nuestra solución), mediante el uso del módulo "random" aportado por los profesores, formando además la lista de candidatos (todos aquellos no incluidos en la lista de seleccionados). A continuación, calcularemos el aporte de cada elemento de la lista de seleccionados aplicando el **uso de la factorización de los cálculos**, mientras haya posibilidad de mejora y no hayamos llegado al número límite de iteraciones; ordenando dicha lista mediante el uso de **dos listas auxiliares**, que nos servirán más adelante para trabajar sobre ellas sin tener que "pisar" los datos que se encuentran en nuestra lista de seleccionados y aportes. La parte más interesante del código comienza con el segundo bucle, cuando mientras no hayamos explorado todo el vecindario y no hayamos obtenido mejora; **en primer lugar**, barajaremos nuestra lista de no seleccionados para evitar la selección secuencial por aportes de los elementos de la lista de no seleccionados, aplicando así mayor grado de aleatoriedad a nuestra solución (ya que como hemos podido comprobar, la aleatoriedad es una buena opción en algunos casos); **en segundo lugar,** tendremos que calcular el aporte de cada elemento de los no seleccionados a la lista solución, quitando de nuestra lista de seleccionados el elemento a cambiar para no añadir, además, su coste a este otro nodo que estamos comprobando (en caso de no obtener un elemento que mejore dicho coste, devolveríamos este elemento a la lista de seleccionados y comprobaríamos el siguiente elemento de la lista de seleccionados). Finalmente, si al explorar todos los elementos seleccionados, no encontramos ninguno capaz de mejorar el coste de la solución, terminaríamos la ejecución del algoritmo.

A continuación, mostraré los resultados de desviación y tiempos de ejecución obtenidos finalmente tras la correcta implementación:

| Algoritmo Búsqueda Local Primer Mejor | | | |
|---|---|---|---|
| **Caso** | **Coste obtenido (Semilla=1234567890)** | **Desv** | **Tiempo** |
| GKD-c_11_n500_m50 | 19586 | 0,01 | 0,15 |
| GKD-c_12_n500_m50 | 19360,1 | 0,00 | 0,15 |
| GKD-c_13_n500_m50 | 19366,1 | 0,00 | 0,15 |
| GKD-c_14_n500_m50 | 19458,6 | 0,00 | 0,15 |
| GKD-c_15_n500_m50 | 19422,1 | 0,00 | 0,13 |
| GKD-c_16_n500_m50 | 19680,2 | 0,00 | 0,15 |
| GKD-c_17_n500_m50 | 19327,8 | 0,02 | 0,15 |
| GKD-c_18_n500_m50 | 19460 | 0,01 | 0,15 |
| GKD-c_19_n500_m50 | 19477,3 | 0,00 | 0,15 |
| GKD-c_20_n500_m50 | 19604,8 | 0,00 | 0,15 |
| MDG-b_1_n500_m50 | 760070 | 2,31 | 0,15 |
| MDG-b_2_n500_m50 | 755987 | 3,07 | 0,15 |
| MDG-b_3_n500_m50 | 762350 | 1,86 | 0,15 |
| MDG-b_4_n500_m50 | 760507 | 1,92 | 0,15 |
| MDG-b_5_n500_m50 | 758756 | 2,17 | 0,15 |
| MDG-b_6_n500_m50 | 761662 | 1,74 | 0,15 |
| MDG-b_7_n500_m50 | 761555 | 2,02 | 0,15 |
| MDG-b_8_n500_m50 | 762629 | 2,12 | 0,15 |
| MDG-b_9_n500_m50 | 761850 | 1,67 | 0,15 |
| MDG-b_10_n500_m50 | 763019 | 1,54 | 0,15 |
| MDG-a_31_n2000_m200 | 113011 | 0,99 | 9,00 |
| MDG-a_32_n2000_m200 | 112814 | 1,12 | 9,00 |
| MDG-a_33_n2000_m200 | 112832 | 1,13 | 9,00 |
| MDG-a_34_n2000_m200 | 112853 | 1,18 | 9,00 |
| MDG-a_35_n2000_m200 | 113196 | 0,86 | 9,00 |
| MDG-a_36_n2000_m200 | 112863 | 1,22 | 9,00 |
| MDG-a_37_n2000_m200 | 113165 | 0,92 | 9,00 |
| MDG-a_38_n2000_m200 | 113102 | 1,12 | 9,00 |
| MDG-a_39_n2000_m200 | 112560 | 1,44 | 9,00 |
| MDG-a_40_n2000_m200 | 113416 | 0,68 | 9,00 |

**Media Desv:** **1,04**

**Media Tiempo:** **3,10**

| Algoritmo | Desv | Tiempo |
|---|---|---|
| Greedy : | **1,37** | **203,33** |
| Búsqueda Local : | **1,04** | **3,10** |

Como podemos comprobar, tanto la desviación como el tiempo de ejecución, en promedio, son considerablemente mejores que en el *enfoque Greedy*; en mayor medida, esto se ve debido a los cambios que comentaba al comienzo de la explicación de este enfoque.

Esta mejora se ha visto motivada principalmente, por el cambio de uso de estructuras de datos, como son el paso del acceso a través de índices a vectores, en comparación con el recorrido realizado por los iteradores ya implementados en la stl a través de listas, junto a la correcta implementación de la factorización de los aportes a la solución, reduciendo significativamente el tiempo de ejecución de los cálculos a cambio de un decremento insignificante de la eficacia del algoritmo.

En una última instancia, traté de implementar el código mediante un vector de parejas, capaz de almacenar tanto los índices de los elementos como los aportes de cada uno de ellos a la solución, aunque en vista del incremento de complejidad producido y la falta de tiempo, no pude experimentar mucho más con ellos, aunque es probable que trate de trabajar con ellos más adelante, ya que considero que la combinación de la estructura de datos *list* junto con estructuras tipo *pairs* podría optimizar aún más los tiempos obtenidos, y no solo eso, sino que con el suficiente tiempo y análisis de estas estructuras, se podría facilitar la comprensión del problema y de operaciones como son la ordenación, borrado o incluso la inserción.

# BIBLIOGRAFÍA:


- Guión de la práctica:
https://pradogrado1920.ugr.es/pluginfile.php/395011/mod_resource/content/16/Guion%20P1a%20LocalGreedy%20MDP%20MHs%202019-20.pdf

- Seminario 2 de la asignatura:
https://pradogrado1920.ugr.es/pluginfile.php/395047/mod_resource/content/9/Sem02-Problemas-BusquedaLocal-MHs-19-20-v2.pdf

- Algoritmos Voraces Iterativos:
http://sdmatull.blogspot.com/2012/03/algoritmos-voraces-iterativos-para-el.html