Metaheurísticas(2019-2020)
Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Propuesta de metaheurística personalizada para el Problema de la Máxima Diversidad (MDP)

Alberto Jesús Durán López
DNI: 54142189-M
albduranlopez@gmail.com

Grupo Jueves, 17:30 - 19:30

27 de junio de 2020

Índice

1. Introducción

Durante el transcurso de la asignatura se ha trabajado con el problema de la máxima diversidad *(Max Diversity Problem)*.
En particular, en esta práctica estudiaremos técnicas de búsqueda basadas en poblaciones para la resolución del problema en cuestión.
Comentaremos todos los pasos y problemas encontrados, detallando minuciosamente todos los detalles y soluciones a los mismos.

Además, se incorporarán tablas para mostrar los resultados de todas las ejecuciones y gráficas para contrastar los modelos (optimización, costes y desviación).

2. Problema de la máxima Diversidad

El problema de la máxima diversidad (*maximum diversity problem*, MDP) es un problema de optimización combinatoria consistente en seleccionar un subconjunto de m elementos ($|M| = m$) de un conjunto inicial N de n elementos (con $n > m$) de forma que se maximice la diversidad entre los elementos escogidos.

El **MDP** se puede formular como:

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

$$\text{Sujeto a: } \sum_{i=1}^{n} x_i = m \quad \text{con } x_i = \{0,1\},\ i = 1, ..., n \quad \text{donde:}$$

- x es una solución al problema que consiste en un vector binario que indica los m elementos seleccionados.
- d_{ij} es la distancia existente entre los elementos i y j

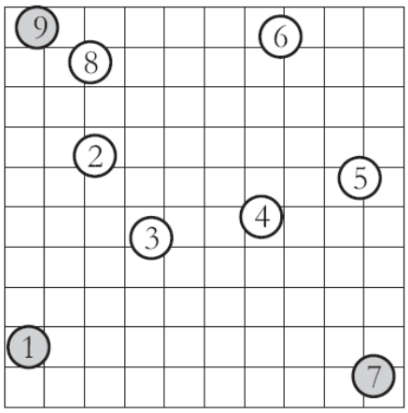


Figura 2.1: MDP-Maximum Diversity Problem

2.1. Datos y casos considerados

Las ejecuciones se han realizado en un ordenador `Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, 16GB RAM, 512 SSD`.
Se utilizarán **30 casos** seleccionados de varios de los conjuntos de instancias disponible en la MDPLIB, 10 pertenecientes al grupo `GKD` con distancias Euclideas, n=500 y m=50 *(GKD-c_11_n500_m50 a GKD-c_20_n500_m50)*, 10 del grupo `MDG` con distancias reales en [0,1000], n=500 y m=50 *(MDG-b_1_n500_m50 a MDG-b_10_n500_m50)*; y otras 10 del grupo `MDG` con distancias enteras en 0,10, n=2000 y m=200 *(MDG-a_31_n2000_m200 a MDG-a_40_n2000_m200)*.

Para la realización de la práctica usaremos el lenguaje de programación `C++` ya que se deben probar muchos ejemplos y la ejecución es más rápida al ser un lenguaje de programación compilado.
Todos los archivos se han compilado con la opción de optimización `-O2` y, además, la semilla usada para todas las ejecuciones ha sido 54142189, pudiéndose ésta indicar en la ejecución del archivo en cuestión.

2.2. Descripción de la Función Objetivo

Todos los algoritmos implementados hacen uso de la misma función objetivo, la comentada en el punto (2) anterior y cuyo pseudocódigo mostramos a continuación. Bien es cierto que en el caso de la *Búsqueda Local* se tiene que usar codificación basada en números enteros, sin embargo, dicha función se ve mínimamente modificada.

Datos: ContribucionIndep(*integer ind, vector sel, matriz distancias)*
inicio
  $suma \leftarrow 0$;
  para j *in* sel **hacer**
    $suma \leftarrow sel[j] \cdot (distancias[ind][j] + suma)$;
  fin
  devolver $suma$
fin

Datos: CosteEstimado(*vector sel, matriz distancias)*
inicio
  $suma \leftarrow 0$;
  para i *in* $|sel|$ **hacer**
    si $sel[i] = true$ **entonces**
      $suma \leftarrow ContribucionIndep(i, sel, distancias) + suma$;
    fin
  fin
  devolver $suma/2$
fin

Algoritmo 1: Evaluar solución

3. Metaheurística original propuesta

Los juegos clásicos que todos conocemos como el buscaminas, tetris o pacman, entre otros, se caracterizan por hacer uso de un tablero o, mejor dicho, un plano finito para su representación.
Esto quiere decir que estos juegos funcionan en un recinto acotado, no pudiendo las diferentes componentes del juego salir de éste.

Por otro lado, existen otros juegos como el *snake* que hacen uso de un tablero cuya superficie es un toro llano. En ésta, cuando la serpiente sale por arriba, entra por abajo y al igual ocurre con los lados, si sale por la derecha, entrará por la izquierda y viceversa. Es decir, podemos establecer una identificación con los bordes de forma que aplicando transformaciones topológicas podemos obtener un toroide tal y como se muestra en la siguiente imagen:

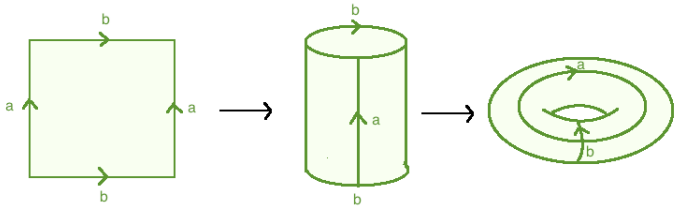


Figura 3.1: Pegado de un toro

Puede además dirigirse al siguiente enlace para ver de forma animada como se lleva a cabo este pegado.

Como podemos imaginar, el toro llano o toroide no es la única superficie que se puede construir identificando lados. De hecho, estas superficies admiten una clasificación que comentamos a continuación:

- Una superficie es **no orientable** si existe al menos una curva cerrada simple contenida que es homeomorfa a una banda de Möbius. Ejemplos:

  1. `Banda de Möbius`: pegado

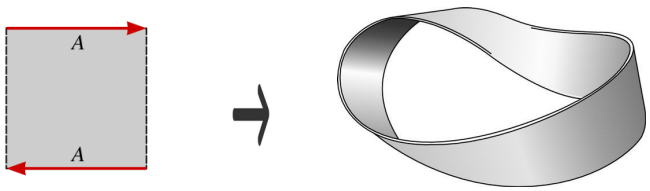


Figura 3.2: Pegado de una banda de Möbius

  2. `Plano Proyectivo`: Quizás las siguientes superficies necesiten algo más de visión espacial ya que el pegado no es tan trivial.

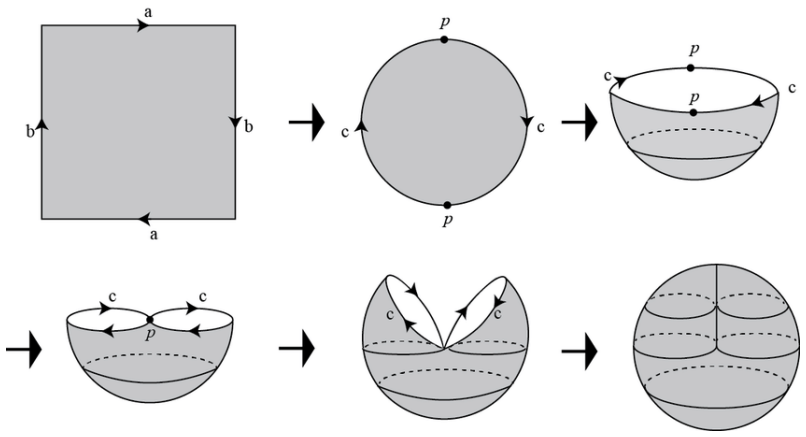


Figura 3.3: Pegado de un Plano Proyectivo

3. `Botella de Klein`: Enlace

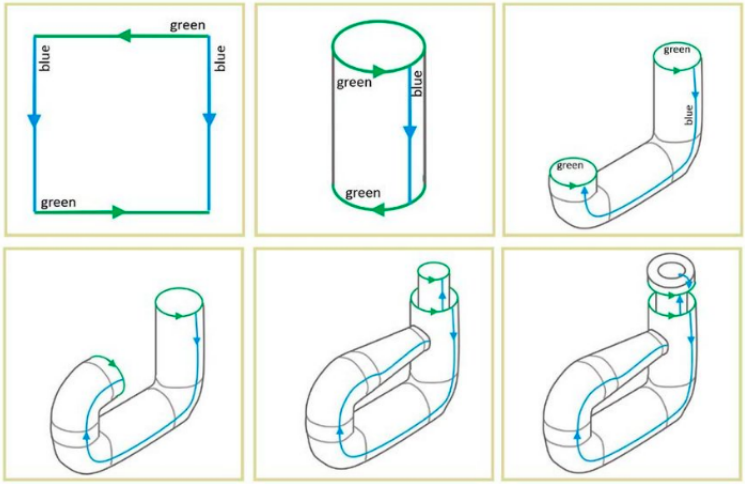


Figura 3.4: Pegado de una Botella de Klein

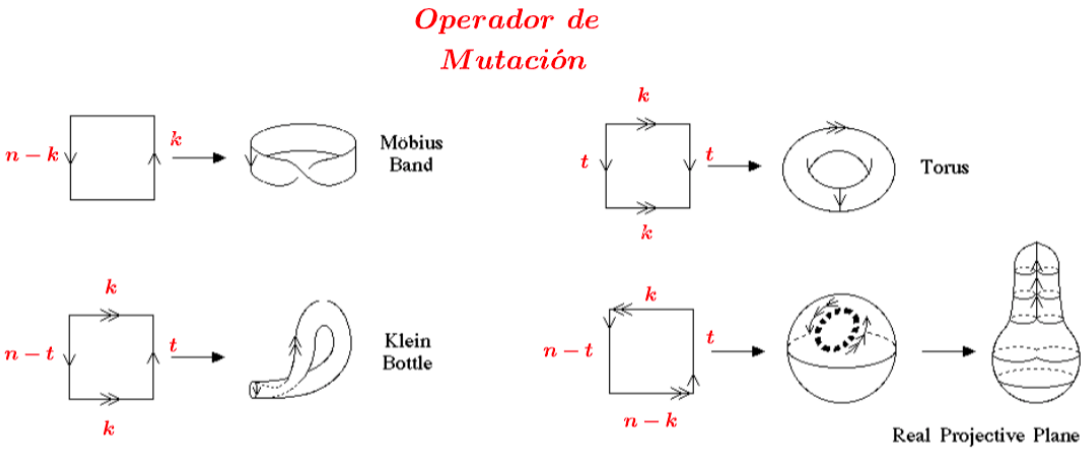
- En caso contrario, la superficie será **orientable**. Ejemplos: Toro llano 3.1, cilindro, suma conexa de k-toros...etc

3.1. Adaptación de la propuesta al problema MDP

Recordando nuestro problema MDP, teníamos que escoger un subconjunto de m elementos de un conjunto inicial de **n** elementos de forma que se maximice la diversidad entre los elementos escogidos. A este conjunto se le suele denominar *sel* (seleccionados) y en una primera versión implementada está creado de forma aleatoria.
En la sección anterior hemos visto como a partir de identificaciones en los bordes de un cuadrado podemos dar lugar a diferentes superficies, por ello, he tomado diferentes soluciones (generadas aleatoriamente) que viven en cada una de las superficies para ver cómo se comportan.
Las características que rigen el comportamiento de nuestro algoritmo son las siguientes:

- `Movimiento`: La operación que simula cada identificación será una mutación. ¿Cómo haremos esto? Relacionándolos de forma que identificaciones en el mismo sentido serán mutaciones en el mismo caracter/posición, mientras que mutaciones en

sentidos opuestos serán mutaciones en las posiciones k y $n-k$, respectivamente. (n es el tamaño de la población).



A modo de ejemplo, lo explicaremos con el famoso juego del *Snake* en la superficie de Toro llano cuyo modelo es el que ha seguido nuestra implementación.

Cuando nuestra serpiente sale por la parte superior, se muta el carácter k, igual que si sale por la parte inferior, ya que en un toro las partes superior e inferior están identificadas. Cuando la serpiente sale por la izquierda o derecha, se muta otro caracter t, ya que ambos lados también están identificados pero son distintos al anterior.

Con esto conseguimos entornos amplios y variados, cada uno caracterizado por tener una exploración de una región distinta.

- `Nutrición`: Siguiendo con las reglas del *Snake*, en todo momento hay un pixel en el tablero que hace referencia a 'comida' que nuestra serpiente puede coger para crecer. En nuestro programa se simulará lo mismo haciendo uso de una distribución uniforme en el intervalo [0,1]. Cuando cada serpiente haya obtenido un número de alimento igual a $\frac{n}{3}$, nuestra solución será lo suficientemente capaz para tener descendencia y dar lugar a una nueva solución en la que se ha producido una mutación brusca (para dar lugar a hijos diferentes en gran medida a los padres y así evitar óptimos locales).

  Los valores de la Distribución Uniforme se han elegido de acuerdo a numerosas pruebas para que las soluciones no crezcan en exceso (ya que a mayor número de soluciones se gastan más evaluaciones de la función objetivo produciendo así una menor convergencia).

3.2. Mejoras tenidas en cuenta en esta primera versión

1. `Mejora en la representación`: Para todos los algoritmos menos para la *Búsqueda Local*, se ha usado una codificación basada en números binarios. Sin embargo, dejaremos la explicación para las siguientes secciones, donde se explicará con detalle en cada algoritmo.

   En relación con la estructura de datos usada, tras varias pruebas se ha optado por la modularización en una clase denominada *Superficie* en la que se le ha de indicar en el constructor los índices referentes a la mutación de cada superficie.

2. Para favorecer la convergencia se ha tenido en cuenta el factor elitista. En cada superficie, si la mejor solución del estado anterior no sobrevive, sustituye directamente la peor solución del estado siguiente. ¿Por qué únicamente la mejor? Para evitar así la convergencia prematura y explorar el máximo de entornos posibles.

3. `Modelo Distribuido. Modelo isla para añadir un operador de viaje`: Tenemos soluciones en diferentes superficies que han ido evolucionando en paralelo pero de forma independiente, explorando así distintas regiones del espacio ya que el operador de mutación aplicado es distinto en cada una de ellas.

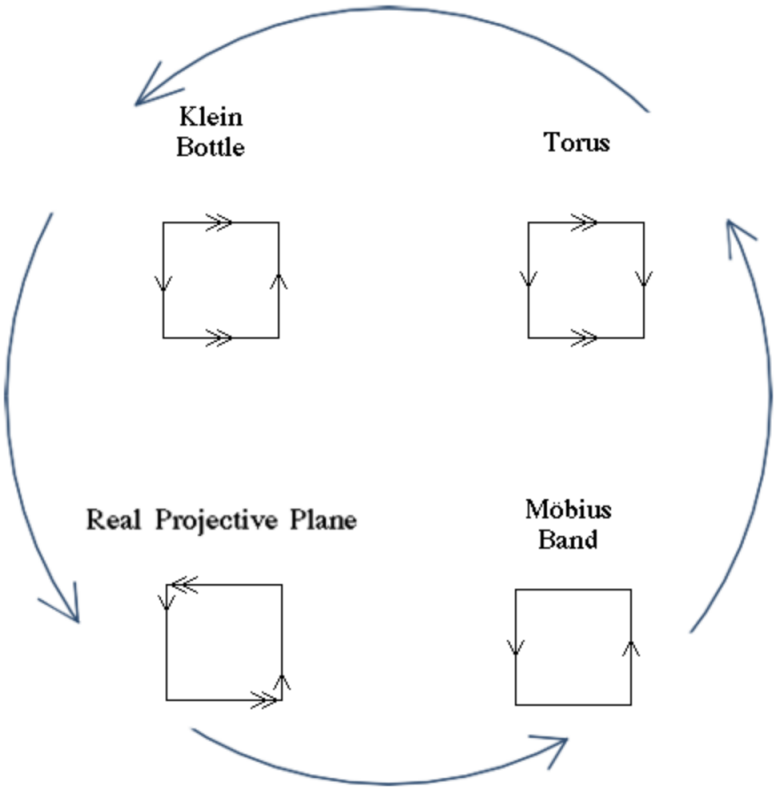


Figura 3.5: Modelo Distribuido

   - Establecemos un operador de viaje en el que la mejor solución de cada Superficie podrá viajar a otra y seguir con el mismo proceso de crecimiento descrito hasta aquí pero con aplicándole un nuevo operador de mutación.
   - Se sigue una estructura de intercomunicación en forma de Anillo ya que el sentido de flujo es único.

- En una primera versión implementada la solución pasaba a otra superficie sin copiarse. Sin embargo, los resultados empeoraban ya que la solución era muy buena en el entorno explorado anterior, pero no tanto cuando se aplicaba el operador de mutación de la nueva superficie. Por ello, el viaje consistirá en una copia de la mejor solución de una superficie a otra para que así nuestra mejor solución pueda seguir explorándose en diferentes superficies.
- Este modelo distribuido proporciona una búsqueda más efectiva que la evolución de una gran población en el que todas las soluciones coexistieran.

4. Las soluciones iniciales están formadas aleatoriamente, por ello, a esta aleatoriedad se han añadido soluciones de calidad obtenidas a partir del algoritmo de *Búsqueda Tabú*. Tendremos un 50 % de soluciones obtenidas de forma aleatoria y otro 50 % de soluciones de la *B. Tabú*. Éste, se introducirá en las siguientes secciones y con él, favorecemos el equilibrio *«Exploración - Explotación»*

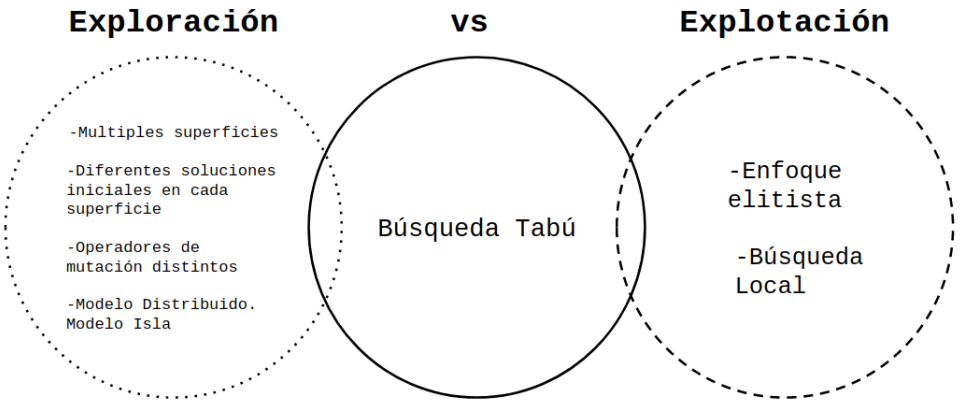


Figura 3.6: Mejoras propuestas

4. «Exploración-Explotación» Multi-Entorno

Al tener tantas superficies caracterizadas por tener mutaciones distintas, me ha parecido adecuado llamar a esta propuesta *«Explotación-Explotación» Multi-Entorno*.
El pseudocódigo asociado es el que se muestra a continuación y que explicaremos brevemente:

- Primeramente, creamos las diferentes superficies *Botella de Klein, Plano proyectivo, Cinta de Möbius y Toro llano*. El constructor de la clase se encargará de asignar mutaciones diferentes a cada una de ellas.
- En segundo lugar, las distintas soluciones que habitan en cada superficie se crearán de forma aleatoria llamando a la función **SolucionAleatoria**.
- A continuación, se lleva a cabo el bucle principal en el que cada solución se mueve libremente por la superficie, coge comida (*nutrirse*) y si la solución crece hasta $\frac{n}{2}$ de su tamaño original es capaz de mudar la piel y dar lugar a nuevas soluciones.
- Por último, dadas dos superficies, la mejor solución de una de ellas viajará a otra (se copiará), pasando así a modificar el operador de mutación que se le aplica (modelo distribuido-isla)

```
Datos: MundoSuperficie
inicio
    Superficies ← {Botella de Klein, Plano proyectivo,
                   Cinta de Möbius, Toro llano } ;

     para s ∈ Superficies[i], i = 1, ..., |Superficies| hacer
        s ← SolucionAleatoria(n,m) ;
    fin
    mientras eval < n_eval hacer
        para s ∈ Superficies[i], i = 1, ..., |Superficies| hacer
            movimiento ;
            nutrirse ;
            mudar_piel ;
        fin
        si se dan las condiciones para viajar entonces
            para s, s' = Superficies[i], Superficies[i + 1]
             hacer
                s viaja a s';
            fin
        fin
    fin
fin
```

Algoritmo 2: Mundo Superficie

5. Búsqueda Local

El algoritmo de *Búsqueda Local* usado para nuestro algoritmo se corresponde con el implementado en la primera práctica, por ello, conviene recordar como funciona:

- Función de intercambio, Int(sel,i,j). El parámetro `i` indica el índice del elemento que se eliminará de la solución y el j por cual se sustituirá. Por tanto, las opciones para `i` serán `m` (los m elementos seleccionados) y las opciones para `j` serán `n-m` (los elementos disponibles en *no seleccionados*), por tanto, el entorno estará formado por `m·(n-m)` elementos.

Datos: EvaluaVecinos
inicio
  $mejora \leftarrow True$;
  $eval \leftarrow 0$;

  mientras *eval<100.000 and mejora* **hacer**
    . . .

    $salir \leftarrow$ False, $c \leftarrow 0$;
    mientras $c < n - m$ *and not salir* **hacer**
      Genera elemento válido j ;
      si $Contrib(s_j) > Contrib(s_i)$ **entonces**
        Int(sel, i, j) ;
        $salir \leftarrow$ True;
        $coste \leftarrow coste + contribNueva - contribAntigua$
      fin
      $c \leftarrow c + 1$;
      $eval \leftarrow eval + 1$;
    fin
  fin
fin

Algoritmo 3: EvaluaVecinos

- En cada iteración, podemos calcular la diferencia de costes entre las dos soluciones recalculando todas las distancias de la función objetivo, sin embargo, esto no es necesario ya que como únicamente añadimos y quitamos 1 elemento, basta con sumar y restar la distancia del nuevo y del viejo elemento al resto de elementos seleccionados, respectivamente. Además, combinamos la factorización del coste con el cálculo de la contribución de los elementos para mejorar aún más la eficiencia.

  Si el intercambio es favorable, es decir, si el coste de la nueva solución (sumando las distancias del nuevo elemento y restando las del elemento anterior) es mayor que la anterior, aceptamos el intercambio. En caso contrario, rechazamos.

- Repetiremos este proceso hasta que se realicen N evaluaciones de la función objetivo o cuando no encuentre mejora en el entorno.

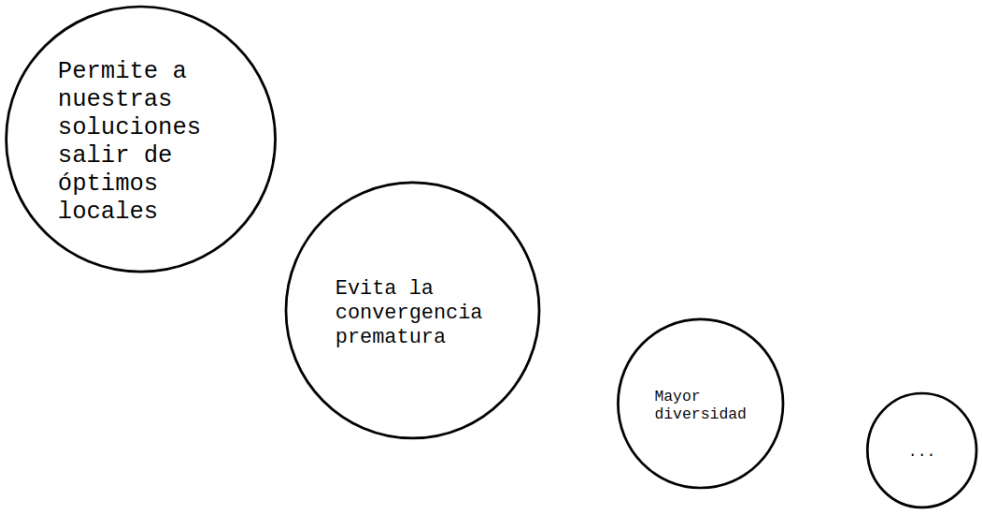
5.1. Hibridación de nuestro problema con Búsqueda Local

Ahora bien, se explicará la forma de hibridación escogida según los resultados obtenidos en los algoritmos meméticos de la práctica 2. En estos, había que programar varias versiones, variando entre aplicar Búsqueda Local a toda la población o una parte de ella. Adicionalmente, se realizó otra versión explorando los $\frac{n}{2}$ peores cromosomas de la población obteniendo incluso mejores resultados que explorar toda la población o una mejor parte de ella.
Por esto, se han realizado pruebas aplicando BL sobre toda la población, mejores soluciones y peores soluciones, obteniendo también mejores resultados aplicando la hibridación sobre los $\frac{n}{2}$ peores soluciones.
El pseudocódigo anterior no se ve modificado a excepción de la siguiente parte:

Datos: MundoSuperficie
inicio
. . .
si *se dan las condiciones para viajar* **entonces**
para $s, s' = Superficies[i], Superficies[i+1]$ **hacer**
aplicar_ BL a los $\frac{n}{2}$ *peores de la población de s* ;
s viaja a s';
fin
fin
fin

Algoritmo 4: Hibridación

Parados en este punto, nos preguntamos: ¿qué ventajas nos ofrece aplicar BL sobre la peor parte de la población?



6. Búsqueda Tabú

La *Búsqueda Tabú* es una metaheurística que utiliza un procedimiento de búsqueda heurística local para explorar el espacio de la solución que pertenece a un óptimo local. Uno de los principales características de este método es el uso de memoria adaptativa, que hace que el comportamiento de búsqueda sea más flexible.Se usarán dos listas:

- `freq`: Almacena el número de veces que el elemento s_i ha sido seleccionado en construcciones previas donde max_freq representa el máximo valor de $freq[i]$ $\forall i \in 0, ..., n$
- `quality`: En esta lista se almacena la media de las soluciones previas en las que el elemento s_i ha sido seleccionado. Por otro lado, max_q hace referencia al máximo valor de $quality[i]$ $\forall i$, $i \in 0, ..., n$

El objetivo es el de usar estar dos listas para formar la solución tomando los elementos más atractivos del total de disponibles:

- S: Elementos totales, *sel*: seleccionados, *S-sel*: Elementos disponibles pero no seleccionados
- s_center: devuelve el elemento que se corresponde con el centro de gravedad del conjunto pasado por argumento y β y δ sirven para incidir en el peso de los valores explicados anteriormente.

Para ello, en cada iteración se modificará la construcción de la solución evaluando la atractividad de cada elemento de *S-sel*. Esto se llevará a cabo dependiendo de los valores que tomen las listas anteriores, favoreciendo los elementos con bajas frecuencias y alta calidad.
Mostramos a continuación el pseudocódigo asociado a la función principal del algoritmo:

Datos: BusquedaTabu
inicio
  $freq[i] = quality[i] = 0 \ \forall s_i \in S$;
  para $eval < n\_eval$ **hacer**
    $sel \leftarrow \emptyset$;
    $s_c = s\_center(S)$;
    mientras $|sel| < m$ **hacer**
      para $s_i \in S - sel$ **hacer**
        $d'(s_i, s_c) \leftarrow d(s_i, s_c) - \beta d(s_i, s_c) \frac{freq[i]}{max\_freq} + \delta d(s_i, s_c) \frac{quality[i]}{max\_q}$;
      fin
      Tomar i^* *tal que* $d'(s_{i^*}, s_c) = max\{d'(s_i, s_c)\}$;
      $freq[i^*] \leftarrow freq[i^*] + 1$;
      $sel \leftarrow sel \cup s_{i^*}$;
      $S \leftarrow S - s_{i^*}$;
      $s_c \leftarrow s\_center(sel)$;
    fin
    $z \leftarrow CosteSolucion(sel)$;
    para $s_i \in sel$ **hacer**
      $quality[i] \leftarrow \frac{quality[i](freq[i]-1)+z}{freq[i]}$;
    fin
  fin
fin

Algoritmo 5: Búsqueda Tabú

7. Resultados Obtenidos

Instancia	Mejor Coste	Superficies	Desv	Superf. + BL	Desv	Superficies BTabú+BL	Desv
GKD-c_11_n500_m50	19587,13	19567,2	0,10	19585,7	0,01	19585,80	0,01
GKD-c_12_n500_m50	19360,24	19333,9	0,14	19360,2	0,00	19359,70	0,00
GKD-c_13_n500_m50	19366,7	19327,8	0,20	19366,7	0,00	19357,80	0,05
GKD-c_14_n500_m50	19458,56	19443,3	0,08	19458,6	0,00	19458,60	0,00
GKD-c_15_n500_m50	19422,15	19381,5	0,21	19422,1	0,00	19413,10	0,05
GKD-c_16_n500_m50	19680,21	19638,2	0,21	19680,2	0,00	19670,40	0,05
GKD-c_17_n500_m50	19331,39	19285,4	0,24	19331,4	0,00	19322,30	0,05
GKD-c_18_n500_m50	19461,39	19405,3	0,29	19461,4	0,00	19446,70	0,08
GKD-c_19_n500_m50	19477,39	19435,1	0,22	19477,3	0,00	19473,50	0,02
GKD-c_20_n500_m50	19604,84	19585,6	0,10	19604,8	0,00	19601,80	0,02
MDG-b_1_n500_m50	778030,62	752733	3,25	769661	1,08	771034	0,90
MDG-b_2_n500_m50	779963,69	763458	2,12	768273	1,50	774127	0,75
MDG-b_3_n500_m50	776768,44	757101	2,53	767690	1,17	767466	1,20
MDG-b_4_n500_m50	775394,62	748318	3,49	763243	1,57	767486	1,02
MDG-b_5_n500_m50	775611,06	750814	3,20	762755	1,66	769415	0,80
MDG-b_6_n500_m50	775153,69	749496	3,31	761953	1,70	767798	0,95
MDG-b_7_n500_m50	777232,87	761057	2,08	772318	0,63	774728	0,32
MDG-b_8_n500_m50	779168,75	753989	3,23	767355	1,52	766385	1,64
MDG-b_9_n500_m50	774802,19	760592	1,83	764790	1,29	768081	0,87
MDG-b_10_n500_m50	774961,31	757519	2,25	768620	0,82	769306	0,73
MDG-a_31_n2000_m200	114139	110748	2,97	112187	1,71	112624	1,33
MDG-a_32_n2000_m200	114092	111284	2,46	112526	1,37	112603	1,31
MDG-a_33_n2000_m200	114124	111338	2,44	112490	1,43	112406	1,51
MDG-a_34_n2000_m200	114203	110854	2,93	112226	1,73	112955	1,09
MDG-a_35_n2000_m200	114180	110458	3,26	112359	1,59	112565	1,41
MDG-a_36_n2000_m200	114252	110865	2,96	112363	1,65	112350	1,66
MDG-a_37_n2000_m200	114213	110675	3,10	112435	1,56	112793	1,24
MDG-a_38_n2000_m200	114378	111600	2,43	112532	1,61	112493	1,65
MDG-a_39_n2000_m200	114201	111032	2,77	112431	1,55	112575	1,42
MDG-a_40_n2000_m200	114191	111558	2,31	112434	1,54	113053	1,00

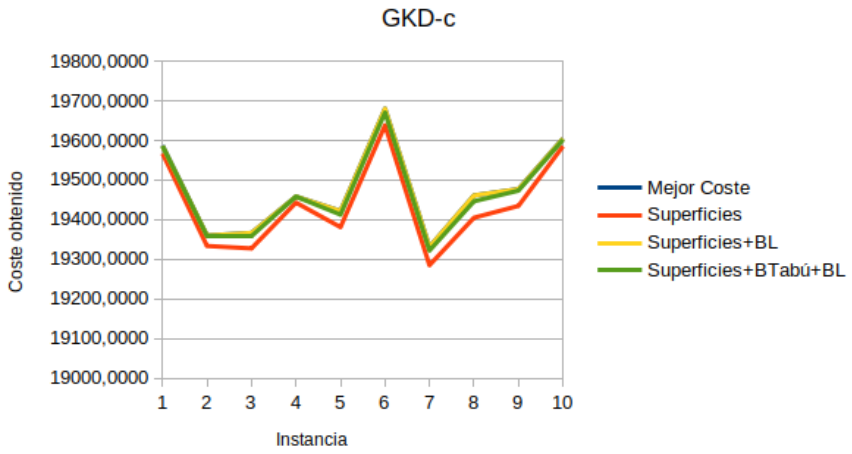
Tabla 7.1: Resultados Generales

Algoritmo	*Desv*	*Tiempo*
$Superficies$	$1,89$	$309,14$
$Superficies + BL$	$0,96$	$39,38$
$Superficies + B.Tabú{+}BL$	$0,77$	$289,34$

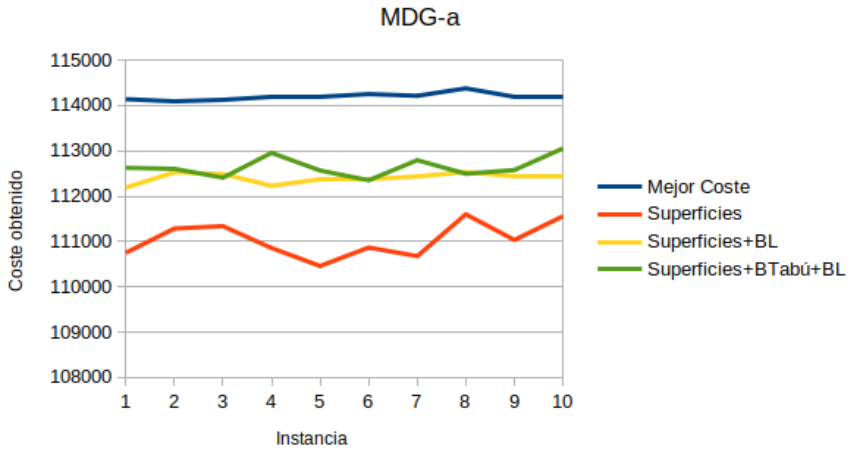
7.1. Análisis de resultados

Tomamos 10 instancias pertenecientes al grupo `GKD` con distancias Euclideas, 10 del grupo `MDG` con distancias reales y otras 10 del grupo `MDG` con distancias enteras. Mostramos los resultados obtenidos en 3 gráficas para reflejar mejor las diferencias:

- `GKD-c`: Conjunto de datos, referentes al grupo de distancias Euclídeas con $n = 500$, $m = 50$, en el que mejor resultados se han obtenido. Se llega prácticamente al óptimo en todas las instancias. Bien es cierto que la versión implementada sin *Búsqueda Tabú* y sin BL se encuentra por debajo del resto, que se superponen con el mejor coste obtenido. Podemos afirmar que hay convergencia con las tres versiones en todas las instancias y, por ello, la desviación obtenida es prácticamente nula.



- `MDG-a`: Conjunto de datos correspondiente al grupo de distancias enteras y tamaño de $n = 2000$ y $m = 200$.

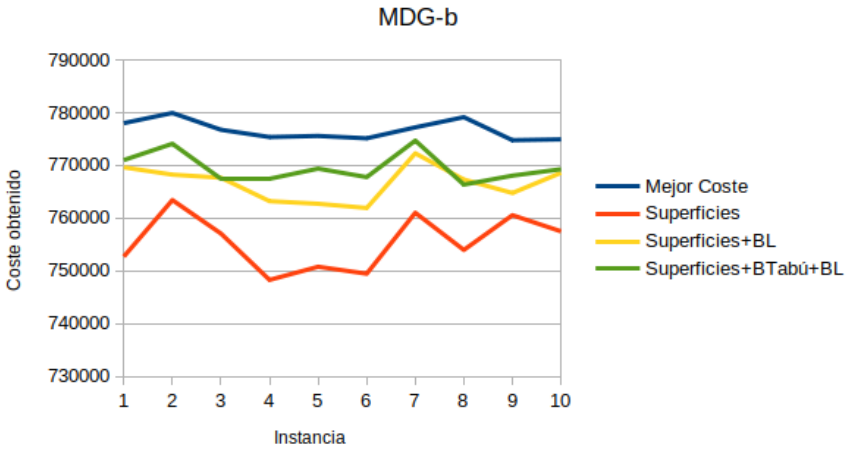


Se observa claramente que el perfil rojo referente a la versión original (sin hibridar) se encuentra por debajo de las otras dos implementaciones. Esto no quiere decir que dicho algoritmo sea malo, de hecho, mejora a todos los algoritmos genéticos de la práctica 2.

En relación con las versiones hibridadas (perfil amarillo y verde), vemos que en todas las instancias menos en 3 de ellas, hibridar el algoritmo con BL y *Búsqueda*

Tabú mejora en gran medida. Esto era de esperar ya que a la hora de crear las distintas soluciones aleatorias, se han añadido algunas de 'calidad' por lo que no dependemos tanto de la aleatoriedad inicial de las soluciones.

- `MDG-b`: Conjunto de datos correspondiente al grupo de distancias reales y tamaño $n = 500$ y $m = 50$.



Destacamos los siguientes aspectos:

- Al igual que ocurre en el grupo de instancias anterior, ambas versiones hibridadas mejoran bastante respecto al algoritmo original debido en cierto modo a las técnicas introducidas para evitar óptimos locales.
- En relación con las versiones hibridadas, el claro ganador vuelve a ser la versión de *Búsqueda Tabú + BL*

7.2. Resultados de todos los algoritmos estudiados

Mostramos a continuación un tabla a modo de resumen general de todos los algoritmos programados en la asignatura, separándolos en práctica 1, 2, 3 y proyecto final de teoría:

Algoritmo	*Desv*	*Tiempo*
Greedy	1,37	0,46
BL	1,07	0,49
$AGG-posición$	2,34	205,13
$AGG-uniforme$	2,53	190,37
$AGE-posición$	2,07	109,06
$AGE-uniforme$	1,42	315,41
$AM-(10,1,0)$	0,74	8,05
$AM-(10,0,1)$	0,89	65,92
$AM-(10,1,0mej)$	0,74	59,24
ES	1,15	0,17
BMB	0,72	0,24
ILS	0,55	0,16
ILS-ES	0,70	1,78
$Superficies$	1,89	309,14
$Superficies+BL$	0,96	39,38
$Superficies+B.Tabú+BL$	0,77	289,34

Como podemos comprobar, los resultados obtenidos en nuestros algoritmos de la entrega final de teoría son bastante prometedores. Si se tuviera que destacar un algoritmo en especial, sería la *Búsqueda Local* y todas sus versiones hibridadas.

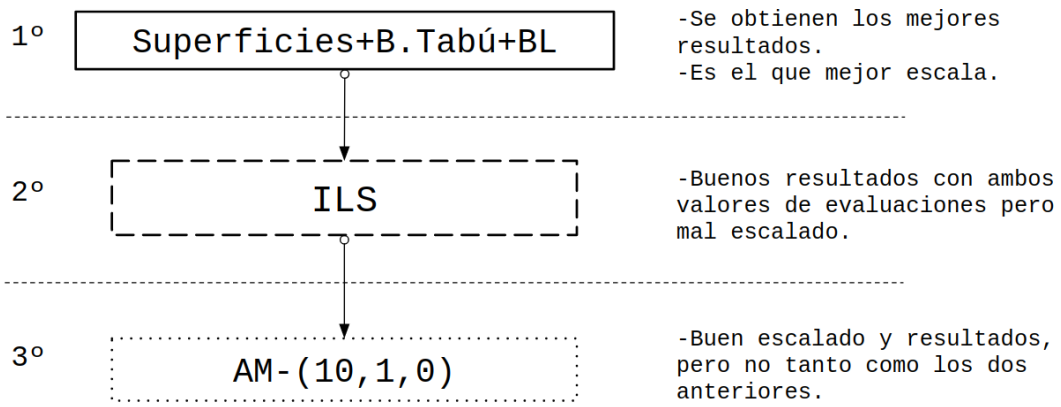
- En comparación con los algoritmos genéticos, vemos que la versión *Superficies* (sin hibridar) mejora a todos los algoritmos genéticos a excepción del Algoritmo Genético Estacionario (*AGE*) en su versión uniforme.
- Sin embargo, en la versión hibridada con *Búsqueda Local*, vemos que tanto los algoritmos meméticos como los programados en esta entrega muestran unas desviación muy parecida.
- Los mejores resultados se han obtenido en la práctica 3, tanto en eficacia como en eficiencia, en concreto *Búsqueda Local Reiterada* (*ILS*), que como bien se comentó se trata de un algoritmo basado en trayectorias múltiples que usa el criterio del mejor como aceptación.

8. Análisis de escalado

Como bien se ha comentado en el punto anterior, los algoritmos híbridos con BL muestran un comportamiento general y no podemos establecer un claro ganador ya que la diferencia de desviación entre unos algoritmos y otros es muy baja.
Para todas estas ejecuciones, el criterio de parada era común, alcanzar 100.000 evaluaciones de la función objetivo. Por ello, realizamos una nueva ejecución de *(AM-(10,1.0)), ILS y Superficies+BTabú+BL* pero estableciendo el criterio de parada a 400.000 evaluaciones.
Mostramos una tabla a modo de resumen de los resultados obtenidos (El estudio con todos las ejecuciones de las instancias se puede ver en el archivo *Estudio.ods*, situado en la carpeta *resultados*):

Algoritmo	$Desv, 100k$ *eval*	$Desv, 400k$ *eval*
$AM-(10,1{,}0)$	$0,74$	$0,60$
ILS	0,55	$0,54$
$Superficies+B.Tab$ú$+BL$	0,77	0,46

Observando la tabla anterior y, para nuestro asombro, vemos que tras 400.000 evaluaciones, el algoritmo *Superficies* es el que mejores resultados obtiene, superando incluso al *Algoritmo Memético* y *Búsqueda Local Reiterada* de las prácticas 2 y 3, respectivamente.

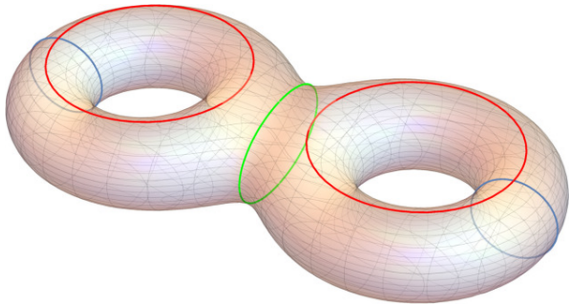


9. Futuros escenarios

El conjunto inicial de superficies en nuestra implementación es de únicamente 4 y, a pesar de ello, los resultados obtenidos son bastante prometedores.
Se podría tomar un mayor número de superficies combinadas con el modelo distribuido en forma de isla, así, el número de entornos explorados serían aún mayor. Éstas superficies podrían además formarse a partir de sumas conexas (uniones) de las superficies anteriores. Destacamos lo siguiente:

- La suma conexa de k-planos proyectivos será una superficie orientable mientras que la suma conexa de k-toros será orientable.

  En la siguiente imagen podemos ver la representación de un 2-toro que podemos perfectamente utilizar como superficie para nuestra propuesta.



  Sin embargo... ¿Cómo podemos llevar la representación de este 2-toro de forma rápida y sencilla a un tablero?

- Para contestar a la pregunta anterior, se ha realizado un programa adicional para simular el movimiento de una solución en un 2-toro y así ver cómo se comportarían las identificaciones en esta supercicie. Éste código se ha entregado junto al proyecto en 2 versiones, una en terminal y otra con interfaz gráfica, en el directorio *snake/*

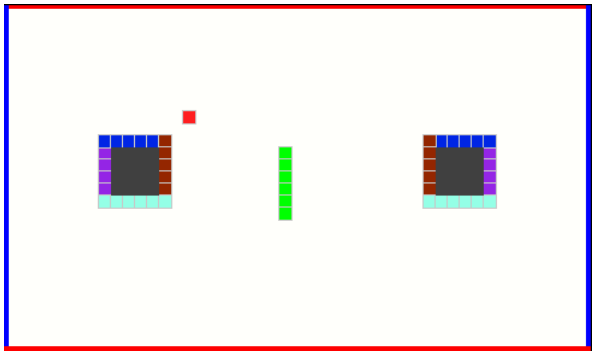


Figura 9.1: snake/snake2-toro.gif

A simple vista, podemos pensar que ambas figuras no son iguales ya que no se conservan las distancias (el recorrido por los arcos grises y rojos de la figura superior son distintos, mientras que esta distancia en la figura inferior no se tiene en cuenta), es decir, ambas figuras no tienen las misma geometría. Sin embargo, topológicamente podemos afirmar que son equivalentes ya que mediante deformaciones podemos obtener una a partir de la otra.
Observando la última figura, vemos que en total tenemos 6 identificaciones luego nuestra solución está caracterizada por tener un entorno en el que 6 caracteres se mutarán, pero... ¿Tener un entorno con mayor número de mutaciones supondría unos mejores resultados?

Referencias

-Seminario 1
-Seminario 2
-Seminario 3
-Seminario 4
-Práctica para evaluar teoría