Práctica 4: Metaheurística NBA Draft

3º CSI
Problema del Agrupamiento con Restricciones (PAR)

Metaheurística propia

David Erena Casado 26055001A davidерena@correo.ugr.es
Grupo 1 (Miercoles 17:30 – 19:30)

Indice:

Problema del Agrupamiento por Restricciones – PAR:

El agrupamiento o análisis de clusters o clustering en inglés, persigue la clasificación de objetos de acuerdo a posibles similitudes entre ellos.

El problema del PAR consiste en una generalización del agrupamiento clásico. Permite incorporar al proceso de agrupamiento un nuevo tipo de información: las restricciones. Dado un conjunto de datos X con n instancias, el problema consiste en encontrar una partición $C=\{c_1,\ldots,c_{1k}\}$ del mismo que minimice la desviación general y cumpla con las restricciones de instancia existentes en el conjunto R. Dada una pareja de instancias, se establece una restricción de tipo Must-Link (ML) si deben pertenecer al mismo grupo y de tipo Cannot-Link (CL) si no pueden pertenecer al mismo grupo.

A partir de ahora, consideraremos n como el número de instancias/datos y k como el número de clústers que consideramos para el problema.

Representación de soluciones e implementación de la función objetivo:

En cuanto a la forma de representar una solución, he decidido hacerlo mediante una clase con los siguientes elementos para cada objeto:

S1.- Un vector donde la posición i indica el cluster al que se asigna el dato i.

S2.- Un vector de vectores donde el vector i almacena los índices de los datos que pertenecen al cluster i.

Centroides.- Centroides usados en el cálculo de la función objetivo – Necesario para factorizar el cálculo de f en vecinos.

f.- Valor de la función objetivo.

Aunque a la hora de mostrar la solución uso S1, ya que el resto de elementos no aporta información útil en este aspecto.

La razón es que acceder a los datos de un cluster es una tarea que se va a realizar muchas veces a lo largo de la ejecución, y si solo trabajásemos con S1 tendríamos que recorrer el vector entero para poder guardar los datos pertenecientes al cluster, mientras que usando S2 no necesitamos esto ya que lo tenemos actualizado en el momento.
A su vez, S1 es necesaria ya que, aparte de para mostrar la solución, para operaciones lógicas (como ver si dos elementos pertenecen al mismo cluster) se complica con S2 (recorrer S2 cada vez que comprobemos algo), y con S1 es inmediato.

Se pierde un poco en espacio y un poco en velocidad al tener que tener ambos actualizados, pero con lo que ahorramos en las operaciones es una mejora.

_____ Funciones referentes a la función objetivo: _____

calcularLambda :

Se usa una vez al principio del programa para calcular cuánto penalizan las infracciones cometidas.
Se calcula dividiendo la mayor distancia entre dos datos del conjunto (siendo la distancia entre dos datos el módulo del vector resta de estos) entre el número de restricciones existentes.

calcularCentroide :

Calcula el centroide asociado a un cluster, realizando la media aritmética de las características de los datos que pertenezcan a este.

vectorCentroides :

Calcula los centroides de todos los clusters, usando la función calcularCentroide.

Ci :

Calcula la distancia intracluster, es decir, la media aritmética de distancias entre cada elemento de un cluster y su centroide (módulo de la resta de estos dos vectores).

1 Total = 0
2 Para cada elemento del cluster
  2.1 distancia = Modulo(centroide – elemento)
  2.2 Total += distancia
3 Return Total/NumElementosCluster

C :

Calcula la media aritmética de los Ci's de cada cluster.

1 Total = 0
2 Desde i=0…NumClusters
  2.1 Total += Ci(Cluster n.º i)
3 Devolver Total/NumClusters

violaRestriccion :

Nos dice si una solución viola una restricción de la matriz de restricciones, es decir, si dos datos dados tenían una relación ML no están en el mismo cluster, o tenían una relación CL y comparten clúster.

infeasibility :

Devuelve el número de restricciones infringidas en una solución. Recorre el triángulo superior de la matriz (sin contar la diagonal principal) ya que es simétrica, y aplica violaRestriccion a cada posición i,j.

1 resultado = 0
2 Desde i=0… NumFilas
  2.1 Desde j=i+1… NumColumnas
    2.1.1 resultado += violaRestriccion(i, j, matrizRestricciones, solucion)
3 Devolver resultado

funcionObjetivo :

Calcula la bondad de una solución, dada por la expresión:
funcionObjetivo = C + (infeasibility * LAMBDA)

restaVector :

Resta dos vectores:
$$v-u=(v_1-u_1, v_2-u_2, \ldots, v_n-u_n)$$

moduloVector :

Calcula el módulo de un vector:
$$|v|=\sqrt{(v_1{}^2+v_2{}^2+\ldots+v_n{}^2)}$$

borrarPorValor :

Borra un elemento de un vector, buscando por valor hasta que alguno coincida con el dado. Para ello, recorre el vector desde el inicio y desde el final hasta llegar a la mitad de este, y en caso de encontrar el elemento lo borra y termina.

GenerarSoluciónAleatoria :

Asignamos números aleatorios entre 0 y numClusters-1 a cada posición de S1.
Repetimos esto hasta que se consiga una solución válida (ningún clúster vacío).
Tras esto, generamos S2 y evaluamos la solución.

Metaheurística NBA Draft:

Motivación:

En la National Basketball Association (siglas NBA), liga de baloncesto profesional en Estados Unidos, se sigue el siguiente proceso para cada temporada (de forma simplificada ya que ciertos detalles no juegan un papel en la metaheurística que se trata):

1. Cada equipo juega 82 partidos de temporada regular. Existen 30 equipos.
2. Los 16 mejores equipos se clasifican para los playoffs, de donde sale el campeón de la NBA.
3. Se realiza el sorteo del draft, donde se decide qué equipo (de los 14 no clasificados para playoffs) posee cada una de las 3 primeras elecciones del draft. El resto de elecciones se otorgan en función del n.º de partidos ganados – perdidos de cada equipo.
4. Se celebra el draft.
5. Empieza una nueva temporada.

Draft: Evento donde cada equipo de la NBA elige a un jugador de universidad / extranjero para que se una a él. En un escenario sin traspasos, cada equipo posee una elección por ronda (actualmente son dos rondas).

Al observar este ciclo, vi una clara relación entre la temporada y el draft y los algoritmos de corte genético que hemos visto en la asignatura, especialmente por parte del draft. Sobre los playoffs, ya existen metaheurísticas que los tratan, y para mi idea, no son esenciales, al igual con otros detalles (por ejemplo, las conferencias que existen y que no he mencionado), por lo que los obviaré.

Resumen:

Tenemos *N* equipos, y jugaremos *T* temporadas.
En cada temporada, cada equipo juega *P* partidos contra cada otro equipo de la liga. Al final, se clasifican según el número de partidos ganados (si existe un empate, se desempatará por valor de la función objetivo).
Para el draft, se asignan las elecciones de forma inversamente proporcional al rendimiento de cada equipo. Tras esto, se generan *N* jugadores a draftear, y se ordenan según su valor de la función objetivo.
Después, se realiza el draft, es decir, se cruzan los equipos con los jugadores del draft según el orden de elección.
Para finalizar, los equipos entrenan de cara a la siguiente temporada y actualizamos la mejor solución encontrada hasta el momento.
Esto constituye una temporada completa.

Pseudocódigo del algoritmo:

1. Generamos *N* soluciones aleatorias.
2. Mientras *temporadasJugadas* < *numTemporadas:*
    1. Jugamos temporada regular.
    2. Obtenemos la clasificación de los equipos.
    3. Asignamos una elección a cada equipo de forma inversa a la clasificación.
    4. Generamos *N* jugadores a draftear, y los ordenamos de menor a mayor f.
    5. Cruzamos cada equipo con su drafteado.
    6. Aplicamos BL a cada equipo.
    7. Comparamos el mejor equipo con la mejor solución encontrada, y actualizamos acorde a ello.

Generar solución aleatoria:

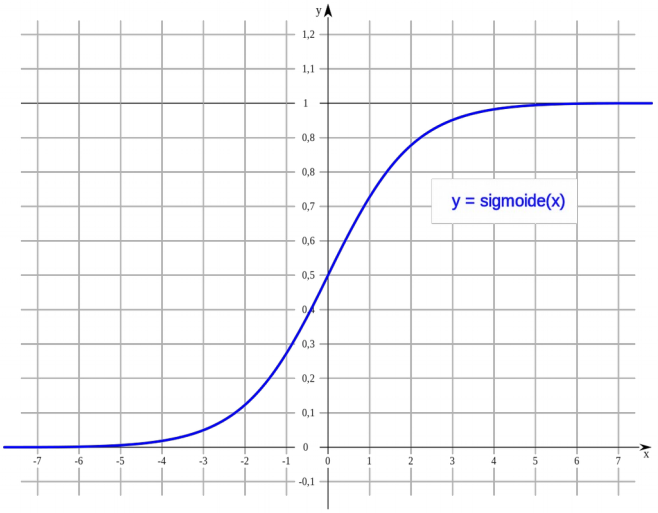
Realizaremos un bucle que nos generará una solución aleatoria en cada iteración de esta, generando n números aleatorios entre 0 y k-1, de forma que acabaremos con una solución al problema. Llevaremos una cuenta del número de componentes de cada cluster para saber si esta es factible o no. En caso de no ser factible, realizaremos otra iteración y generaremos otra solución.

Temporada regular:

Cada equipo juega *P* partidos contra cada equipo.

Cada partido se decide según la probabilidad dada por la función

$$sigmoide(x,y)=\frac{1}{1+e^{\frac{x}{y}}}$$



Busco que el resultado de un partido no sea determinista completamente, y esta función nos da justo eso. Así, si el valor de f para una solución *x* es a y f para otra *y* es b, realizaríamos sigmoide(a-b, min{a,b}). Uso el mínimo de a y b con el objetivo de que la pendiente que realiza la función hacia su asíntota se suavice un poco, dando más oportunidades al peor equipo (solo represento sigmoide(x) para ver de forma clara el valor que nos da la función). Además, si ambos equipos tienen f igual, habrá 50% de probabilidad para cada uno.

Clasificación de los equipos:

Obtenemos un vector que nos indica la clasificación de los equipos, acorde a los partidos ganados de cada uno, y en caso de empate, mayor f.

Pseudocódigo:

1. aux = partidosGanados
2. Para i=0..NumEquipos
    1. Para j=0..NumEquipos
        1. indiceMejorEquipo = j => aux[j] es el máximo de aux con mayor f posible
    2. clasificación[i] = j
    3. aux[j] = -1

Cruce de equipo con el drafteado:

Usamos el cruce por segmento fijo usando en la práctica 2.
Generamos inicio (*ini*) y longitud (*long*) del segmento como n.º aleatorios entre 0 y n-1.
Luego, copiamos el segmento de longitud *long* empezado en *ini* de un padre en el hijo (Mientras no se hayan copiado *long* elementos, copiamos el elemento *i* del padre en el hijo (comenzando *i* en *ini*, y *i=i+1 mod* vector.size)

Búsqueda Local:

Partiendo de una solución dada, comenzamos el siguiente bucle:

1. Si no hemos generado el entorno virtual de la solución en la que nos encontramos (la mejor hasta el momento), lo generamos (de acuerdo a lo explicado en la exploración de entorno).
2. Obtenemos un elemento del entorno (el último de la lista, en mi caso, pero como se baraja al generarse dicha lista, no tiene mucha relevancia).
3. Generamos el vecino asociado a dicho elemento, y lo evaluamos.
4. Si el valor obtenido es menor que el mínimo, guardamos este valor como el mínimo y la solución del que surge como la mejor solución obtenida) y volveríamos al principio del bucle.
5. Si no lo es y ya no nos quedan elementos en el entorno, o hemos realizado 10000 evaluaciones en el bucle, salimos de este.

Exploración del entorno: Representaremos un posible vecino como un par (i,j) que indicará un cambio del dato i al cluster j.
Para explorar el entorno, crearemos una lista de estos pares para no tener que guardar una gran cantidad de vectores, de los cuales puede que muchos sean inútiles, es decir, que siempre encontremos un vecino mejor mucho antes de que llegue su turno de ser evaluado.
A su vez, solo incluiremos pares que acaben generando vecinos en los que se haga un cambio y además nos lleve a una solución factible.

Generación de vecino: Como ya hemos hecho todas las comprobaciones al generar el entorno de exploración, no tendremos que hacerlas al generar un vecino. Por lo tanto, solo tenemos que realizar una copia de la solución actual cambiando el valor que nos indica el par para S1, y borrar en el vector del cluster e incluirlo en otro para S2.

Justificación:

Es un algoritmo de fácil comprensión en su estructura básica, como suelen ser los algortimos genéticos.
El draft proporciona una forma diferente de explorar el entorno de soluciones de una forma diferente a las que hemos visto en la asignatura, ya que en este caso realizamos cruces con una población diferente.
En un principio, sí que falta explotación, ya que no realizamos nada por ella. Sin embargo, en mi idea original, la incluyo tras el draft, con la explicación de que cada equipo se entrena para la siguiente temporada.
Además, es un algoritmo muy modular: tienes varias componentes claramente separadas entre ellas, y eso permite poder ajustar cada una por separado, de forma que se pueden obtener muchas variantes.

Hibridación con Búsqueda Local:

Nota: Usaré N=30, T=5 y P=2 para los experimentos de ahora en adelante. Hasta que no se especifique lo contrario, usaré un orden de draft de forma inversa a la clasificación obtenida.

Incluyo búsqueda local después de realizarse el draft y haber cruzado los equipos con su jugador drafteado.
Los resultados sin BL y con BL pueden verse aquí:

10% de restricciones:

	IRIS				ECOLI				RAND				NEWTHYROID			
	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T
SIN BL	1,888646	453,2	4,763318	0,01981264	45,55834	1778,6	93,27738	0,08353794	2,754844	457,6	6,050302	0,01987848	13,10032	1109,8	53,68716	0,03493284
BL A TODOS	0,669305	0	0,669305	10,21012	20,69252	96,2	23,27404	358,2484	0,715599	0	0,715599	9,026502	13,835	6	14,0544	35,03368

20% de restricciones:

	IRIS				ECOLI				RAND				NEWTHYROID			
	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T
SIN BL	1,877658	931,2	4,829662	0,02348008	45,35306	3560,6	93,11772	0,1090504	2,672078	939,4	6,053164	0,02268692	13,06748	2252,8	54,25238	0,04637016
BL A TODOS	0,669305	0	0,669305	11,97936	21,73878	140,6	23,6257	443,6684	0,715599	0	0,715599	11,57296	14,2871	0	14,2871	45,41178

Como era de esperar, sin un componente que realice la explotación de las soluciones, nunca llegamos a una buena solución en un tiempo muy corto (el resto de operaciones en el algoritmo no son tan costosas en tiempo de ejecución). Lo único que hacemos es cruzar soluciones entre ellas sin ningún objetivo.
Al incluir búsqueda local a todos los equipos, obtenemos muy buenos resultados, pero el tiempo de ejecución para conjuntos grandes, como Ecoli, es extremadamente largo (7 minutos aproximadamente con 20% de restricciones).
Por lo tanto, vemos que es esencial para el algoritmo tener un componente que explote las soluciones ya que sin él es imposible que converja, pero puede que nos beneficiemos de no excedernos tanto en ese frente, ya que no podemos descuidar del tiempo de ejecución.

Mejoras sobre el diseño/comportamiento de la MHe:

El primer problema, como ha sido expresado en el apartado anterior, es que la búsqueda local nos condiciona demasiado en tiempo. Por tanto, hay que reducir el número de soluciones a las que aplicamos BL.
He decidido probar a aplicar BL a los *N/3* peores, *N/3* mejores equipos y finalmente, *N/2* mejores equipos.
Opté por *N/3* en un principio, ya que uso 30 equipos (el mismo número que en la NBA).
Los resultados son los siguientes:

10% de restricciones:

	IRIS				ECOLI				RAND				NEWTHYROID			
	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T
BL A TODOS	0,669305	0	0,669305	10,21012	20,69252	96,2	23,27404	358,2484	0,715599	0	0,715599	9,026502	13,835	6	14,0544	35,03368
BL A N/3 PEORES	0,669305	0	0,669305	4,046458	22,45056	107,8	25,34476	118,123	0,715599	0	0,715599	3,841296	13,835	6	14,0544	14,21304
BL A N/3 MEJORES	0,669305	0	0,669305	3,226988	21,27072	81,6	23,46026	113,714	0,715599	0	0,715599	3,166558	13,835	6	14,0544	11,59064
BL A N/2 MEJORES	0,669305	0	0,669305	4,861894	20,45992	104,8	23,2719	174,029	0,715599	0	0,715599	4,758064	13,835	6	14,0544	17,29154

20% de restricciones:

	IRIS				ECOLI				RAND				NEWTHYROID			
	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T
BL A TODOS	0,669305	0	0,669305	11,97936	21,73878	140,6	23,6257	443,6684	0,715599	0	0,715599	11,57296	14,2871	0	14,2871	45,41178
BL A N/3 PEORES	0,669305	0	0,669305	4,76421	22,40538	153,6	24,4678	153,5284	0,715599	0	0,715599	4,434826	14,2871	0	14,2871	17,437
BL A N/3 MEJORES	0,669305	0	0,669305	3,97145	21,98146	122	23,61856	153,0256	0,715599	0	0,715599	3,733838	14,2871	0	14,2871	14,81712
BL A N/2 MEJORES	0,669305	0	0,669305	5,83023	21,80846	126,6	23,5068	211,5736	0,715599	0	0,715599	5,600994	14,2871	0	14,2871	22,29398

Conseguimos el objetivo de reducir el tiempo de ejecución, de hecho, podemos ver que la BL podría considerarse nuestro cuello de botella, ya que el tiempo de ejecución es más o menos proporcional al número de equipos a los que aplicamos búsqueda local.

A su vez, podemos usar una porción con cierta seguridad de su efectividad ya que, dentro de empeorar el valor f conseguido de media, es un aumento muy ligero y no lo considero nocivo.

También vemos que con *N/3* mejores equipos, obtenemos mejores resultados, lo cual se debe a que estamos explotando las soluciones en principio más prometedoras.
Por tanto, a partir de ahora, usaré BL aplicado a los *N/3* mejores equipos ya que nos consigue buenos resultados en un buen tiempo.

IMPORTANTE: A partir de ahora, únicamente usaré el conjunto de datos Ecoli, ya que los otros conjuntos darán el óptimo siempre y los tiempos obtenidos en ellos serán similares a los obtenidos en esta primera prueba.

Como siguiente posible modificación, he pensado en el orden del draft. Es decir, ver si nos conviene unir el peor equipo con el mejor a draftear o si conseguimos mejores soluciones cambiando este orden. Para ello, aparte de los datos obtenidos ahora (que, como se ha especificado antes, usa un orden inverso a la clasificación), también obtengo los datos obtenidos con un orden igual a la clasificación (es decir, el mejor jugador a draftear con el primer equipo y así sucesivamente) y un orden aleatorio.

10% de restricciones:

	NBA DRAFT (DRAFT ALEATORIO) – 10%			
	ECOLI			
	Tasa_C	Tasa_inf	Agr,	T
Ejecución 1	21,2723	104	24,0628	106,042
Ejecución 2	21,8791	64	23,5963	107,422
Ejecución 3	21,2218	90	23,6365	105,02
Ejecución 4	21,7676	68	23,5921	106,368
Ejecución 5	21,3666	90	23,7813	123,995
MEDIA	21,50148	83,2	23,7338	109,7694

	NBA DRAFT (DRAFT PROPORCIONAL) – 10%			
	ECOLI			
	Tasa_C	Tasa_inf	Agr,	T
Ejecución 1	21,6176	87	23,9522	116,367
Ejecución 2	20,8041	113	23,8359	113,473
Ejecución 3	19,6126	117	22,7518	104,796
Ejecución 4	21,6309	77	23,6969	105,003
Ejecución 5	19,5624	115	22,6479	123,589
MEDIA	20,64552	101,8	23,37694	112,6456

	ECOLI			
	Tasa_C	Tasa_inf	Agr,	T
DRAFT ALEATORIO	21,50148	83,2	23,7338	109,7694
DRAFT PROPORCIONAL	20,64552	101,8	23,37694	112,6456

20% de restricciones:

	NBA DRAFT (DRAFT ALEATORIO) – 20%			
	ECOLI			
	Tasa_C	Tasa_inf	Agr,	T
Ejecución 1	21,8686	140	23,7468	139,222
Ejecución 2	21,9243	111	23,4137	140,065
Ejecución 3	21,8064	145	23,7516	137,538
Ejecución 4	21,1541	207	23,9314	139,759
Ejecución 5	21,9472	137	23,7855	152,469
MEDIA	21,74012	148	23,7258	141,8106

	NBA DRAFT (DRAFT PROPORCIONAL) – 20%			
	ECOLI			
	Tasa_C	Tasa_inf	Agr,	T
Ejecución 1	21,8697	146	23,8283	137,319
Ejecución 2	22,1528	117	23,7228	137,951
Ejecución 3	21,8479	131	23,6052	138,186
Ejecución 4	21,7085	169	23,9759	138,526
Ejecución 5	21,7639	149	23,7628	138,571
MEDIA	21,86856	142,4	23,779	138,1106

	ECOLI			
	Tasa_C	Tasa_inf	Agr,	T
DRAFT ALEATORIO	21,74012	148	23,7258	141,8106
DRAFT PROPORCIONAL	21,86856	142,4	23,779	138,1106

Como se obtienen resultados similares para ambos con 20% de restricciones pero algo mejor para 10% con draft proporcional (conseguimos bajar de f=23), considero el draft proporcional a la clasificación como la opción a usar a partir de ahora.

Material considerado y manual de uso:

En cuanto al material usado en la implementación, se han usado las funciones de generación de números aleatorios de PRADO (random.h y random.cpp) y las siguientes librerías de C++ : iostream, fstream, string, vector, set, cmath, numeric, algorithm, limits.h, chrono, functional .

En cuanto al uso del código, este viene dividido en 2 carpetas: *BIN* y *FUENTES*. Las acompaña un *makefile*, con lo cual, para compilar el código, solo necesita usar el comando *make* posicionándose en la terminal sobre la carpeta donde se encuentre. Dentro de *FUENTES*, están las carpetas *include* (archivos .h), *obj* (archivos .o) y *src* (archivos .cpp), y en *BIN* se encontrará el ejecutable y una carpeta *data* con los datos para ejecutar el programa.

Una vez compilado, el programa, que se encontrará en la carpeta BIN, se ejecuta con los siguientes argumentos:

```
./BIN/practica4  archivoDeDatos  archivoDeRestricciones  NumeroDeClusters  Semilla
```

Comparación con otros algoritmos:

Vemos cómo rinde la MHe (ya con el rediseño) junto al resto de algoritmos genéticos que hemos realizado durante el curso:

10 % de restricciones:

	ECOLI			
	Tasa_C	Tasa_inf	Agr,	T
AGG-UN	23,67786	143,6	27,53056	22,65584
AGG_SF	26,17966	210	31,81386	22,90906
AGE_UN	21,86638	75,4	23,88936	23,99094
AGE_SF	21,71882	94,2	24,24614	23,71036
AM-1,0,10	21,99724	79,2	24,12216	22,18634
AM-0,1,10	21,9211	81,6	24,1104	22,56036
AM-0,1,10mej	21,42468	111	24,40278	22,62968
NBA DRAFT	20,64552	101,8	23,37694	112,6456

20% de restricciones:

	ECOLI			
	Tasa_C	Tasa_inf	Agr,	T
AGG-UN	22,96608	308	27,09782	29,51196
AGG_SF	24,6265	312,8	28,82262	29,39096
AGE_UN	21,86736	152,4	23,91178	30,79072
AGE_SF	21,95504	163,4	24,147	30,49766
AM-1,0,10	21,96348	180,8	24,38888	29,137
AM-0,1,10	21,69284	191,2	24,25778	29,15254
AM-0,1,10mej	21,84446	175,2	24,19472	29,26158
NBA DRAFT	21,86856	142,4	23,779	138,1106

Obtenemos el mejor valor de f de todos los algoritmos genéticos, pero aún así el algoritmo tarda mucho más que los otros (2 minutos de NBA Draft contra 30 segundos aprox. del resto). Esto podría remediarse limitando aún más la búsqueda local, limitando así la explotación, o reduciendo el número de equipos, que limitaría la exploración de soluciones.