Práctica 1: Búsquedas con trayectorias simples

Antonio Álvarez Caballero 15457968-J
5º Doble Grado Ingeniería Informática y Matemáticas
Grupo de prácticas del Viernes 17:30-19:30
analca3@correo.ugr.es

14 de abril de 2016

Índice

1. Descripción del problema

El problema a resolver es el problema de *Selección de características*.En el ámbito de la *Ciencia de Datos*, la cantidad de datos a evaluar para obtener buenos resultados es excesivamente grande. Esto nos lleva a la siguiente cuestión: ¿Son todos ellos realmente importantes? ¿Podemos establecer dependencias para eliminar los que no nos aportan información relevante? La respuesta es que sí: en muchas ocasiones, no todos los datos son importantes, o no lo son demasiado. Por ello, se intentará filtrar las características relevantes de un conjunto de datos.

La selección de características tiene varias ventajas: se reduce la complejidad del problema, disminuyendo el tiempo de ejecución. También se aumenta la capacidad de generalización puesto que tenemos menos variables que tener en cuenta, además de conseguir resultados más simples y fáciles de entender e interpretar.

Para conseguir este propósito se deben usar técnicas probabilísticas, ya que es un problema *NP-hard*. Una técnica exhaustiva sería totalmente inviable para cualquier caso de búsqueda medianamente grande. Usaremos metaheurísticas para resolver este problema, aunque también podríamos intentar resolverlo utilizando estadísticos (correlación entre características, medidas de separabilidad o basadas en teoría de información o consistencia, etc).

2. Descripción de aplicación de los algoritmos al problema

Los elementos comunes de los algoritmos son:

- Representación de las soluciones: Se representan las soluciones como vectores 1-dimensionales binarios (los llamaremos *bits* para poder hacer uso de términos como *darle la vuelta a un bit*):

$$s = (x_1, x_2, \ldots, x_{n-1}, x_n);\ x_i \in \{True, False\}\ \forall i \in \{1, 2, \ldots, n\}$$

- Función objetivo: La función a maximizar es la tasa de clasificación de los datos de entrada:

$$tasa\_clas = 100 \cdot \frac{instancias\ bien\ clasificadas}{instancias\ totales}$$

- Generación de vecino: La función generadora de vecinos es bien simple. Se toma una solución y se le da la vuelta a uno de sus bits, el cual se escoge aleatoriamente.

```
Tomar una solución
indice = generarAleatorio(0, numero_caracteristicas)
caracteristicas[indice] = not caracteristicas[indice]
```

3. Descripción de la estructura del método de búsqueda

Veamos los esquemas de cada algoritmo en pseudocódigo:

- Búsqueda Local:

```
caracteristicas = generaSolAleatoria()
fin = Falso
mejor_tasa = coste(caracteristicas)
evaluaciones = 0
```

```
Mientras no fin && evaluaciones < MAXIMO_EVALUACIONES
  lista_vecinos = aleatoriza(caracteristicas)
  Para cada caracteristica de lista_vecinos
    Si está activa
      Continuar
    Si evaluaciones >= MAXIMO_EVALUACIONES
      Break

    flip(caracteristicas, indice_caracteristica)
    tasa = coste(caracteristicas)
    flip(caracteristicas, indice_caracteristica)

    si tasa > mejor_tasa:
      mejor_tasa = tasa
      flip(caracteristicas,indice_caracteristica)
      Break
  Si no he terminado debido a un Break:
    fin = True
```

- Enfriamiento Simulado:

```
caracteristicas = generaSolAleatoria()
fin = Falso
mejor_tasa = coste(caracteristicas)
evaluaciones = 0
mejor_solucion = caracteristicas
tasa_actual = mejor_tasa

Inicializar valores de temperatura y demás parámetros

Mientras temperatura >= temperatura_final && evaluaciones < MAXIMO_EVALUACIONES
    vecinos_aceptados = 0
    Para cada vecino en 0..max_vecinos
      Si vecinos_aceptados >= max_aceptados
        Break

      caracteristica = aleatorio()
      flip(caracteristicas, caracteristica)
      nueva_tasa = coste(caracteristicas)

      delta = tasa_actual - nueva_tasa

      Si delta != 0 && (delta < 0 || aleatorio < exp(-delta/temperatura))
        vecinos_aceptados++
        tasa_actual = nueva_tasa
        Si tasa_actual > mejor_tasa
          mejor_tasa = nueva_tasa
          mejor_solucion = caracteristicas
      Si no
        flip(caracteristicas, caracteristica)
```

```
    temperatura = actualizar(temperatura)
```

- Búsqueda Tabú:

```
caracteristicas = generaSolAleatoria()
fin = Falso
mejor_tasa = coste(caracteristicas)
evaluaciones = 0
mejor_solucion = caracteristicas
tasa_actual = mejor_tasa

Inicializar valores de lista tabú y demás parámetros

Mientras evaluaciones < MAXIMO_EVALUACIONES
  tasa_actual = 0
  mejor_caracteristica = -1

  vecindario = generaAleatorio(tamaño_vecindario)

  Para cada vecino en vecindario
    flip(caracteristicas, vecino)
    nueva_tasa = coste(caracteristicas)
    flip(caracteristicas, vecino)

    evaluaciones++

    Si vecino está en lista_tabú
      Si nueva_tasa > mejor_tasa && nueva_tasa > tasa_actual
        tasa_actual = nueva_tasa
        mejor_caracteristica = vecino
    Else Si nueva_tasa > tasa_actual
      tasa_actual = nueva_tasa
      mejor_caracteristica = vecino

    Si evaluaciones >= MAXIMO_EVALUACIONES
      Break

    posicion_lista_tabu = (posicion_lista_tabu + 1) % tamaño_lista_tabu
    lista_tabu[posicion_lista_tabu] = mejor_caracteristica

    Si tasa_actual > mejor_tasa
      mejor_tasa = tasa_actual
      flip(caraceristicas, mejor_caracteristica)
```

- Búsqueda Tabú extendida:

```
caracteristicas = generaSolAleatoria()
fin = Falso
mejor_tasa = coste(caracteristicas)
evaluaciones = 0
```

```
mejor_solucion = caracteristicas
tasa_actual = mejor_tasa
iteraciones_sin_mejora = 0

Inicializar valores de lista tabú y demás parámetros

Mientras evaluaciones < MAXIMO_EVALUACIONES
  tasa_actual = 0
  mejor_caracteristica = -1

  if iteraciones_sin_mejora >= 10:
    iteraciones_sin_mejora = 0
    eleccion = generaAleatorio()
    Si eleccion < 0.25
      caracteristicas = generaSolAleatoria()
    Si eleccion < 0.5
      caracteristicas = mejor_solucion
    Si no
      total_sols = sum(frec)
      caracteristicas = generaVectorFrecuencias()

    eleccion = generaAleatorio()
    Si eleccion < 0.5
      tamaño_lista_tabu *= 1.5
    Si no
      tamaño_lista_tabu /= 1.5
    tamaño_lista_tabu = parteEnteraPorEncima(tamaño_lista_tabu)
    tabu_list = lista(-1,tamaño_lista_tabu)

  vecindario = generaAleatorio(tamaño_vecindario)

  Para cada vecino en vecindario
    flip(caracteristicas, vecino)
    nueva_tasa = coste(caracteristicas)
    flip(caracteristicas, vecino)

    evaluaciones++

    Si vecino está en lista_tabú
      Si nueva_tasa > mejor_tasa && nueva_tasa > tasa_actual
        tasa_actual = nueva_tasa
        mejor_caracteristica = vecino
    Else Si nueva_tasa > tasa_actual
      tasa_actual = nueva_tasa
      mejor_caracteristica = vecino

    Si evaluaciones >= MAXIMO_EVALUACIONES
      Break

    posicion_lista_tabu = (posicion_lista_tabu + 1) % tamaño_lista_tabu
    lista_tabu[posicion_lista_tabu] = mejor_caracteristica
```

```
Si tasa_actual > mejor_tasa
  mejor_tasa = tasa_actual
  flip(caraceristicas, mejor_caracteristica)
  mejor_solucion = caracteristicas
```

4. Descripción del algoritmo de comparación

El algoritmo de comparación es un algoritmo greedy: el *Sequential Forward Selection(SFS)*. La idea es muy simple: se parte del conjunto vacío de características (todos los bits a 0) y se recorren todas las características, evaluando la función de coste. La característica que más mejora ofrezca, se coje. Y se vuelve a empezar. Así hasta que ninguna de las características mejore el coste.

```
caracteristicas = (1,2,...,n)
caracteristicas_seleccionadas = (0,0,...,0,0)
fin = falso
mejor_caracteristica = 0


Mientras mejor_caracteristica != -1
  mejor_tasa = 0
  mejor_caracteristica = -1
  Para cada característica
    tasa = coste(característica)
    Si tasa > mejor_tasa
      mejor_tasa = tasa
      mejor_caracteristica = caracteristica
  Si mejor_caracteristica != -1
    caracteristicas_seleccionadas.añadir(mejor_caracteristica)
```

5. Desarrollo de la práctica

En primer lugar, comentar que las bases de datos han sido modificadas en su estructura (que no en sus datos) para que sean homogéneas. Así, se han puesto todas las clases como numéricas (en Wdbc no lo estaban) y se han colocado en la última columna.

La práctica se ha desarrollado usando el lenguage de programación *Python*, ya que su velocidad de desarrollo es bastante alta. Para intentar lidiar con la lentitud que puede suponer usar un lenguaje interpretado, utilizaremos las librerías *NumPy, SciPy y Scikit-Learn*, que tienen módulos implementados en C (sobre todo *NumPy*) y agilizan bastante los cálculos y el manejo de vectores grandes.

Usaremos alguna funcionalidad directa de estas bibliotecas:

- *NumPy*: Generación de números aleatorios y operaciones rápidas sobre vectores.
- *SciPy*: Lectura de ficheros ARFF de WEKA.
- *Scikit-Learn*: Particionamiento de los datos, tanto las particiones estratificadas de la validación cruzada 5x2 como las de *Leave One Out* para calcular la función de coste. También se ha tomado un clasificador KNN, ya que está implementado usando estructuras de datos complejas como *Ball Tree* y lo hace muy rápido.

Esta elección se ha hecho para poder preocuparme sólo y exclusivamente de la implementación de las metaheurísticas.

Los requisitos para ejecutar mis prácticas son *Python3* (importante que sea la 3), *NumPy*, *SciPy* y *Scikit-Learn*. En mi plataforma (Archlinux) están disponibles desde su gestor de paquetes.

Una vez instalados los paquetes, sólo hay que ejecutar la práctica diciéndole al programa los algoritmos que queremos ejecutar. La semilla aleatoria está fijada dentro del código como 12345678 para no inducir a errores. Veamos algunos ejemplos de llamadas a la práctica. Primero notamos que los algoritmos disponibles son:

- -SFS: Ejecuta el algoritmo greedy SFS.
- -LS: Ejecuta la Local Search.
- -SA: Ejecuta el Simulated Annealing.
- -TS: Ejecuta la Tabu Search.
- -TSext: Ejecuta la Tabu Search extendida.

```
$ python practica1.py -TS
```

Se ejecutará la Tabu Search. Pero no sólo se limita el programa a un algoritmo. Si le pasamos varios, los ejecutará en paralelo con el criterio de ejecutar tantos procesos como el mínimo entre los algoritmos pasados por argumento y el número de CPU's del sistema. Así, cada algoritmo podrá ir a una CPU distinta sin interferir en el rendimiento de ninguno de ellos.

```
$ python practica1.py -SFS -LS -SA
```

Se ejecutarán en paralelo SFS, LS y SA paralelamente. Si se tuvieran dos núcleos, se ejecutarían los dos primeros y el tercero esperaría, para así no lastrar el rendimiento de ambos.

Una vez ejecutado, irán saliendo por pantalla mensajes de este tipo, que proporcionan datos en tiempo real del estado de la ejecución:

```
INFO:__main__:W - TS - Time elapsed: 2265.526112794876.
Score: 98.2394337654. Score out: 95.0877192982 Selected features: 15
```

Este mensaje nos dice todo lo necesario: W es la base de datos (Wdbc), TS el algoritmo, el tiempo transcurrido para esta iteración (recordemos que hay 10), el score de entrenamiento, el score de validación y las características seleccionadas.

6. Experimentos

Como se ha comentado antes, la semilla está fija a 12345678 para no tener problemas de aleatoriedad. El número de evaluaciones máxima de todos los algoritmos es de 15000. Por lo demás, todos los demás parámetros propios de cada algoritmo están tal y como se explica en el guión ($\phi = \mu = 0{,}3$, los valores de vecinos máximos, soluciones máximas aceptadas, etc).

KNN

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	96.12676	96.84211	0.0	0.0	67.77778	65.0	0.0	0.0	64.58333	66.49485	0.0	0.0
Partición 1-2	96.49123	96.83099	0.0	0.0	66.66667	80.55556	0.0	0.0	63.40206	62.5	0.0	0.0
Partición 2-1	96.12676	96.49123	0.0	0.0	66.11111	74.44444	0.0	0.0	64.0625	64.43299	0.0	0.0
Partición 2-2	95.78947	96.12676	0.0	0.0	70.55556	68.88889	0.0	0.0	64.94845	64.58333	0.0	0.0
Partición 3-1	96.47887	95.4386	0.0	0.0	76.11111	71.66667	0.0	0.0	64.0625	63.91753	0.0	0.0
Partición 3-2	96.8421	96.83099	0.0	0.0	66.66667	65.55556	0.0	0.0	63.40206	64.58333	0.0	0.0
Partición 4-1	97.53521	96.14035	0.0	0.0	65.0	66.11111	0.0	0.0	67.1875	64.43299	0.0	0.0
Partición 4-2	93.33333	97.88732	0.0	0.0	70.0	72.77778	0.0	0.0	62.8866	64.58333	0.0	0.0
Partición 5-1	96.47887	96.84211	0.0	0.0	71.66666	71.11111	0.0	0.0	65.10417	65.97938	0.0	0.0
Partición 5-2	97.89473	95.42254	0.0	0.0	65.55555	70.0	0.0	0.0	63.40206	63.54167	0.0	0.0
Media	96.30973	96.48530	0.00000	0.0	68.61111	70.61111	0.00000	0.0	64.30412	64.50494	0.00000	0.0

SFS

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	97.53521	92.2807	83.33333	0.15793	72.22222	66.66667	93.33333	0.50295	80.20833	70.61856	96.76259	2.41429
Partición 1-2	96.84211	94.01408	86.66667	0.12852	77.22222	66.66667	87.77778	0.91927	73.19588	67.1875	97.48201	1.84876
Partición 2-1	95.42254	91.22807	83.33333	0.15797	84.44444	68.88889	85.55556	1.0978	79.16667	69.58763	97.48201	1.83412
Partición 2-2	97.54386	92.60563	86.66667	0.12911	77.77778	61.66667	93.33333	0.49637	74.2268	64.58333	97.48201	1.88374
Partición 3-1	96.12676	92.98246	90.0	0.09982	83.33333	71.66667	87.77778	0.92519	76.5625	68.5567	98.20144	1.30598
Partición 3-2	97.54386	96.47887	86.66667	0.12927	72.22222	60.0	93.33333	0.50101	71.64948	66.14583	98.20144	1.33008
Partición 4-1	98.23944	96.49123	86.66667	0.12883	70.55556	62.22222	91.11111	0.66223	76.04167	69.58763	97.84173	1.55438
Partición 4-2	94.73684	94.3662	90.0	0.10095	80.55556	65.55556	90.0	0.74934	82.98969	73.95833	97.1223	2.16781
Partición 5-1	94.71831	91.92982	90.0	0.09997	78.88889	66.11111	92.22222	0.57865	77.08333	68.04124	97.48201	1.85395
Partición 5-2	98.94737	93.66197	76.66667	0.21516	76.66667	66.66667	90.0	0.74815	82.98969	71.35417	96.40288	2.75023
Media	96.76563	93.60390	86.00000	0.13475	77.38889	65.61111	90.44444	0.71810	77.41140	68.96209	97.44604	1.89433

LS

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	97.88732	95.08772	43.33333	0.03328	69.44444	64.44444	44.44444	0.24336	65.10417	67.01031	47.84173	3.89067
Partición 1-2	98.24561	96.47887	16.66667	0.04433	67.22222	77.77778	44.44444	0.14276	65.46392	61.97917	44.96403	1.37486
Partición 2-1	96.47887	97.54386	40.0	0.04197	72.77778	69.44444	33.33333	0.24853	68.22917	65.46392	53.59712	1.47268
Partición 2-2	98.24561	94.3662	43.33333	0.03995	73.33333	71.11111	46.66667	0.16294	71.64948	64.58333	49.64029	2.0683
Partición 3-1	98.23944	93.33333	43.33333	0.04072	76.66667	67.77778	42.22222	0.18122	67.70833	65.97938	48.92086	3.69346
Partición 3-2	95.78947	97.1831	43.33333	0.04351	70.0	71.11111	51.11111	0.11783	70.10309	64.0625	48.20144	1.98576
Partición 4-1	96.83099	96.14035	33.33333	0.02976	71.66667	70.55556	41.11111	0.11845	68.22917	68.04124	45.68345	2.53283
Partición 4-2	97.19298	96.47887	33.33333	0.04327	75.0	74.44444	47.77778	0.23483	65.97938	65.625	49.28058	2.58812
Partición 5-1	96.47887	96.14035	50.0	0.05204	72.22222	67.22222	50.0	0.16506	67.70833	65.97938	43.88489	1.46301
Partición 5-2	97.89474	94.01408	40.0	0.05042	78.33333	71.11111	41.11111	0.16569	67.01031	64.58333	52.8777	2.22095
Media	97.32839	95.67667	38.66666	0.04193	72.66667	70.50000	44.22222	0.17807	67.71854	65.33076	48.48921	2.32906

SA

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	97.53521	96.49123	40.0	1.11269	75.0	65.0	48.88889	4.18387	70.83333	65.97938	53.59712	52.13983
Partición 1-2	97.89474	96.83099	63.33333	0.92206	71.11111	78.33333	44.44444	4.37269	68.04124	61.45833	46.40288	50.67776
Partición 2-1	97.53521	94.38596	50.0	1.03374	74.44444	72.22222	51.11111	4.0898	67.1875	65.46392	48.20144	60.19112
Partición 2-2	96.49123	95.07042	36.66667	1.16516	75.0	69.44444	55.55556	3.77572	70.61856	63.02083	50.0	45.56738
Partición 3-1	97.1831	96.14035	26.66667	1.22105	76.11111	76.11111	51.11111	4.08054	71.875	65.46392	47.48201	60.50781
Partición 3-2	97.19298	95.42254	40.0	1.12074	78.33333	67.77778	47.77778	4.22231	67.01031	68.22917	48.92086	46.92599
Partición 4-1	98.94366	93.33333	46.66667	1.06667	79.44444	70.55556	54.44444	3.88628	73.95833	69.58763	51.43885	53.92757
Partición 4-2	96.14035	96.83099	30.0	1.19294	72.22222	70.55556	43.33333	4.42988	70.61856	65.625	47.84173	48.29168
Partición 5-1	96.83099	94.38596	46.66667	1.06748	75.0	72.77778	37.77778	4.69352	70.83333	68.04124	49.64029	56.86653
Partición 5-2	98.24561	96.12676	50.0	1.03651	74.44444	72.22222	53.33333	3.9502	70.10309	64.0625	51.79856	43.62812
Media	97.39931	95.50185	43.00000	1.09390	75.11111	71.50000	48.77778	4.16848	70.10792	65.69319	49.53237	51.87238

TS

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	98.23944	95.08772	50.0	24.513	70.0	64.44444	48.88889	32.84348	70.3125	66.49485	53.95683	118.46261
Partición 1-2	97.19298	93.30986	53.33333	23.64954	70.55556	76.66667	62.22222	27.40562	72.16495	61.45833	56.47482	91.90189
Partición 2-1	98.59155	91.22807	63.33333	21.47659	75.0	65.55556	52.22222	31.59482	67.1875	64.43299	53.59712	118.73509
Partición 2-2	95.78947	96.83099	46.66667	25.21243	77.22222	66.66667	58.88889	28.4676	73.19588	67.70833	47.84173	112.23147
Partición 3-1	97.1831	97.19298	40.0	26.31299	73.88889	73.88889	54.44444	30.11558	70.83333	65.97938	48.56115	139.88007
Partición 3-2	97.54386	94.3662	53.33333	23.59961	66.11111	73.33333	60.0	27.77152	65.97938	62.5	48.92086	110.00699
Partición 4-1	96.47887	96.84211	43.33333	25.6046	78.33333	70.55556	50.0	32.45111	71.35417	64.43299	48.56115	139.71234
Partición 4-2	97.54386	95.77465	43.33333	25.69186	69.44444	70.0	38.88889	36.58017	69.07216	68.22917	50.71942	105.65995
Partición 5-1	97.1831	97.19298	46.66667	24.94993	70.0	68.33333	63.33333	26.46016	68.75	63.40206	53.95683	117.80619
Partición 5-2	99.29825	96.47887	53.33333	23.5993	80.55556	74.44444	54.44444	30.48133	72.16495	65.625	48.92086	110.03668
Media	97.50445	95.43044	49.33333	24.46099	73.11111	70.38889	54.33333	30.41714	70.10148	65.02631	51.15108	116.44333

TSExt

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	98.23943	95.08771	50.0	2356.32802	73.33333	66.66666	55.55555	1515.91046	70.3125	65.46391	48.92086	2366.13130
Partición 1-2	98.24561	95.77464	56.66666	2337.36061	73.88888	76.66666	57.77777	1509.71581	69.07216	66.14583	51.79856	2331.48903
Partición 2-1	97.88732	95.08771	60.0	2364.73440	74.44444	70.55555	48.88888	1502.66443	70.83333	65.97938	50.0	2293.80195
Partición 2-2	98.59648	95.77464	36.66666	2332.66297	72.77777	70.0	50.0	1497.51423	70.10309	69.27083	51.07913	2357.94027
Partición 3-1	97.88732	95.78947	6.66666	2345.16758	76.11111	71.66666	47.77777	1495.34883	71.35416	68.04123	51.79856	2279.87537
Partición 3-2	97.89473	95.77464	50.0	2320.83067	72.22222	71.66666	56.66666	1499.41790	71.13401	64.0625	55.39568	2302.68525
Partición 4-1	97.18309	97.54385	30.0	2321.19691	73.88888	64.44444	54.44444	1509.17492	72.91666	61.34020	52.15827	2242.64845
Partición 4-2	99.29824	94.36619	56.66666	2335.17310	67.77777	73.33333	53.33333	1496.48844	71.13401	68.22916	46.76258	2265.25803
Partición 5-1	98.59154	95.08771	43.33333	2277.79662	77.77777	71.66666	54.44444	1514.86820	70.3125	64.94845	48.92086	2235.67741
Partición 5-2	98.94737	93.30985	56.66666	2311.27374	67.77777	74.44444	55.55555	1516.11590	73.19587	65.10416	52.51798	2273.49106
Media	98.20264	95.58739	43.33333	2332.361209	73.58024	70.74073	53.20987	1504.50988	70.79693	65.94238	50.75938	2297.27856

Media

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
KNN	96.30973	96.48530	0.00000	0.0	68.61111	70.61111	0.00000	0.0	64.30412	64.50494	0.00000	0.0
SFS	96.83642	93.56845	86.66667	19.80493	72.77778	63.83333	90.33333	73.72346	77.66216	71.04328	97.94964	181.17828
LS	97.32839	95.67667	38.66666	3.64440	70.77778	71.27778	45.22222	7.85552	68.03318	65.02201	46.54676	48.30436
SA	97.39931	95.50185	43.00000	101.98625	71.44444	71.22222	48.44445	210.64471	69.89691	65.02201	50.39568	914.34243
TS	97.50445	95.43044	49.33333	2206.51355	70.88889	70.22222	54.66667	1395.62846	69.57904	65.23411	51.22302	1961.12589
TSExt	98.20264	95.58739	43.33333	2332.361209	73.58024	70.74073	53.20987	1504.50988	70.79693	65.94238	50.75938	2297.27856

Las conclusiones son claras: La búsqueda tabú es un gran algoritmo, pero en algunos casos se ve superado por otros más sencillos. Bien optimizada puede dar grandes resultados (aún tardando muchísimo más que los demás), pero si uno tiene poco tiempo para desarrollar un algoritmo de optimización, es más viable irse a un Simulated Annealing, por ejemplo, que ofrece resultados muy parecidos tardando muchísimo menos. En algunos casos la simple búsqueda local es bastante efectiva.

La búsqueda tabú extendida es una buena opción sobre la normal: con un poco más de tiempo se consigue mejorar un poco los resultados. Si nos decantamos por una tabú, debería tener memoria a largo plazo, ya que tarda tanto, que al menos optimice lo máximo.

7. Referencias

Las referencias utilizadas han sido:

- *Scikit-Learn*: La propia documentación[1] de la biblioteca.
- *SciPy*: La propia documentación[2] de la biblioteca.

[1] http://scikit-learn.org/stable/modules/classes.html
[2] http://docs.scipy.org/doc/scipy/reference/