Iterated Greedy aplicado al problema de la selección de características en KDD

Alfonso Fernández, Abraham Duarte, Rosa M. Hernández, Ángel Sánchez
Dpto. Ciencias de la Computación
Universidad Rey Juan Carlos
28933 Móstoles
alfonso.fernandez@urjc.es, abraham.duarte@urjc.es, rm.hernandez@alumnos.urjc.es, angel.sanchez@urjc.es

Resumen

La calidad de los modelos extraídos de una base de datos y que representan la información almacenada en la misma, depende en gran medida, de la fase inicial de preprocesamiento de dichos datos. Las principales tareas a realizar en esta fase son dos: por un lado, la limpieza de datos inconsistentes y/o redundantes y, por otro, la reducción del tamaño de la misma (debido a la dificultad que conlleva trabajar con gran cantidad de datos). De las distintas técnicas existentes para esta tarea, hemos centrado nuestra atención en la selección de características. El objetivo de este trabajo es reducir la cantidad de información que caracteriza a cada una de las instancias, disminuyendo el número de atributos de las mismas. Para ello hemos desarrollado un procedimiento basado en una metodología voraz iterativa (*IG*) y los resultados obtenidos han sido comparados con los extraídos de otros trabajos, con el objetivo de comprobar la calidad del método diseñado.

1. Introducción

A medida que el mundo se va globalizando, la cantidad de datos que maneja la sociedad es cada vez mayor. Conforme aumenta el volumen de estos datos, surgen nuevos problemas y preguntas como el qué hacer con tanta información redundante (ya que mucha de ella no se necesita o podemos excluirla), o cuál sería la solución a los tiempos de procesamiento de datos, ya que al tener mucha más cantidad, la carga de trabajo de los ordenadores es mayor, y el tiempo en la terminación de las tareas aumenta.

A partir de un análisis explícito y detallado, la metodología conocida como KDD (*Knowledge Discovery in Databases*), tiene la tarea de inferir información valida a partir de estos datos [3].

Posteriormente, dicha información pasa a manos de la Minería de Datos (*Data Mining* o DM), cuya principal labor es la de desarrollar modelos que se puedan utilizar en diversos campos de la ciencia, la ingeniería o la economía [5].

Sin embargo, la calidad de estos modelos está íntimamente relacionada con la de los datos almacenados, dependiendo de factores como son la existencia de datos erróneos o el gran tamaño de la base de datos.

Por lo tanto urge, como fase inicial del proceso, el preprocesamiento de los datos iniciales, con el fin de elevar la calidad de los mismos.

Basándonos en una metodología iterativa voraz (en inglés *Iterated Greedy IG*), hemos desarrollado un algoritmo (con dos versiones), con el fin de aplicarlo al preprocesamiento de conjuntos de datos. El objetivo es obtener subconjuntos de los mismos, que los representen de manera fiable y que faciliten la obtención de información a los procesos siguientes en la Minería de Datos.

Por lo tanto, en las primeras secciones expondremos las diferentes técnicas para el preprocesamiento de datos, para luego describir el algoritmo *Iterated Greedy* diseñado para este trabajo. Por último, aplicaremos el mismo sobre diferentes conjuntos de datos, comparándolos con resultados del estado del arte y extrayendo las posibles conclusiones.

2. Preprocesamiento en KDD

Las diferentes técnicas de preprocesamiento de las que dispone la Minería de Datos [6] pueden ser enumeradas en los siguientes puntos:

- *Reducción de datos:* obtener un conjunto menor, representativo de los datos originales.

- *Limpieza de datos:* eliminar datos erróneos, incompletos o redundantes.
- *Integración de datos:* fusionar bases de datos concernientes al mismo tipo de problema.
- *Transformación de datos:* realizar modificaciones sintácticas sobre los datos sin que varíe su significado.

De las diferentes técnicas presentadas, nos centraremos en la de reducción de datos. Los modelos extraídos por las técnicas de *Data Mining* pueden llegar a ser difíciles de interpretar si el volumen de datos es demasiado grande. Teniendo en cuenta además que, a mayor cantidad de datos, hay que emplear más cantidad de tiempo para generar los modelos, una de las principales técnicas de preprocesamiento es la reducción del tamaño del conjunto de datos.

Existen diferentes caminos a la hora de disminuir la cantidad de datos inicial disponible. Haciendo una clasificación con las mismas, las podemos agrupar en las siguientes opciones:

- *Selección de Instancias:* se eligen las instancias más representativas del conjunto.
- *Selección de Características:* se eliminan aquellos atributos menos relevantes para la obtención de los modelos.
- *Discretización de Características:* **c**onsiste en cuantificar atributos numéricos continuos.
- *Agrupamiento de Datos:* agrupar las instancias en conjuntos en función de su afinidad o relación.
- *Compactación de Datos:* método similar a la selección de instancias, donde las instancias de la tabla final cuentan con un nuevo atributo, representativo del número de instancias desechadas más próximas a ella.

De entre estas técnicas, se ha desarrollado en este trabajo la selección de características, a describir en la próxima sección.

3. Selección de características

Una de las técnicas más usuales y con mayor éxito en el preprocesamiento de datos es la de selección de características [1,2]. Mediante esta técnica se eliminan del conjunto inicial, aquéllas que representan a determinados atributos de las instancias con menor peso específico en la clasificación definitiva de los mismos. Esto puede deberse, entre otras circunstancias, al hecho de que el papel que representa dicho atributo puede ser representado por otro u otros atributos de dicho objeto, pasando a ser su influencia prácticamente testimonial. De esta forma se consigue un conjunto más reducido de datos que repercute en una reducción en la complejidad tanto en tiempo como en memoria utilizada, a la hora de aplicar la Minería de Datos.

Existen multitud de técnicas a la hora de determinar si una determinada cualidad o característica es relevante para tomar una decisión. Sin ir más lejos, los árboles de decisión constituyen una técnica ampliamente divulgada cuyo objetivo es elegir el atributo más prometedor para llevar a cabo la división en cada nodo interno del mismo, no debiendo seleccionar nunca uno irrelevante y carente de sentido.

De las diferentes formas de clasificación de las técnicas de selección de características, la más extendida quizá sea aquella basada en el mecanismo de selección empleado. Atendiendo a esta clasificación, dos son las alternativas: filtro (*filter*) y envoltura (*wrapper*). El modelo filtro evalúa los atributos de acuerdo a heurísticas considerando características generales de los datos. La segunda técnica utiliza el comportamiento de un algoritmo de clasificación como criterio de evaluación de los atributos.

Otras vías adecuadas para la clasificación de los algoritmos de selección de atributos pueden ser aquellas basadas en las propias características de los algoritmos de selección: medida de calidad del subconjunto seleccionado, la estrategia de búsqueda (que puede ser completa, estocástica o heurística) y la dirección de la búsqueda. Según este último criterio podemos clasificar el procedimiento de selección de características en tres categorías:

- *Búsqueda secuencial constructiva:* En este caso partimos de un conjunto vacío de características al cual vamos adicionando, en cada paso, una nueva hasta alcanzar una cierta condición de parada.
- *Búsqueda secuencial destructiva:* Implementando una estrategia opuesta a la presentada en el punto anterior, partimos de un conjunto completo de características y en cada una de las iteraciones se va descartando

una de ellas hasta alcanzar la condición de parada.

- *Búsqueda aleatoria:* Se sigue un procedimiento aleatorio, que evita caer en óptimos locales.

En nuestro caso implementaremos dos algoritmos que, dentro de una panorámica Voraz Iterativa (en inglés *iterated greedy* IG), implementan procedimientos que combinan técnicas secuenciales constructivas y destructivas. Para ello partimos de un conjunto de instancias (que denominaremos tabla original), caracterizadas por una serie de atributos comunes, y clasificadas en función del valor de los mismos en diferentes clases. En la Tabla 1 podemos observar una base de datos con 6 instancias y dos clases, donde cada instancia (fila) tiene 4 atributos (las cuatro primeras columnas) y pertenece a una clase (última columna).

Tabla 1. Conjunto de seis instancia con 4 atributos

Inst	*Atr. 1*	*Atr. 2*	*Atr. 3*	*Atr. 4*	*Clase*
1	4.6	3.4	1.4	0.3	A
2	6.7	4.1	5.2	0.5	B
3	4.5	2.9	1.8	0.3	A
4	8.0	3.4	1.9	0.2	A
5	7.9	4.6	6.2	0.9	B
6	0.6	3.3	5.2	0.1	A

Nuestro objetivo es construir un conjunto más reducido que el original (que llamaremos tabla reducida), formado por los atributos más representativos de las mismas y con la capacidad de clasificar aquellas nuevas instancias que se puedan generar.

De los diferentes criterios disponibles para clasificar una determinada instancia en una clase u otra, utilizaremos la regla del vecino más cercano. Es decir, la nueva instancia será clasificada en la clase de la instancia perteneciente a la tabla reducida, más cercana de la que se encuentre. En todos los casos, la distancia implementada fue la euclídea, para un espacio de n dimensiones (donde n representa el número de atributos de las instancias presentes en ese determinado momento en la tabla reducida).

Para ponderar la calidad de la tabla reducida obtenida tras la aplicación del algoritmo nos ayudaremos de una función de *fitness f*. Esta función combinará la capacidad de clasificación de la tabla final con la reducción llevada a cabo por el algoritmo, desde el conjunto inicial de características al número final presente en la tabla reducida. La función de *fitness f* será:

$$f = \alpha*(PorcAcierto) + (1-\alpha)*(PorcReducción)$$

donde *PorcAcierto* simboliza el porcentaje de instancias correctamente clasificadas utilizando sólo las características presentes en las instancias de la tabla reducida. Por otro lado, *PorcReducción* se define como uno menos el cociente entre el tamaño de la tabla reducida y el tamaño de la tabla original. El parámetro α es un valor entre 0 y 1, que ponderará el peso que se le da a cada una de estas cualidades de la tabla reducida. En este trabajo, el valor tomado para α fue siempre de 0.5, dándosele la misma importancia a la reducción y al porcentaje de acierto. En función de la aplicación concreta, se podría dar más importancia a cualquiera de las dos magnitudes descritas.

Como ya se comentó con anterioridad, los procedimientos desarrollados pueden enmarcarse dentro de la metodología Voraz Iterativa, que pasaremos a desarrollar en la siguiente sección.

4. Algoritmo voraz iterativo

Una de las metaheurísticas de reciente creación y que más éxitos y adeptos está cosechando, es el algoritmo voraz iterativo (*IG*) [7,8]. Dicho esquema algoritmo destaca por su sencillez a la hora de ser planteado y la calidad de las soluciones obtenidas tras su aplicación. Es fácilmente parametrizable y puede ser adaptado con facilidad a gran cantidad de problemas.

Dicha metodología consta de la combinación de dos fases complementarias, una constructiva y otra destructiva, donde una de ellas sigue una estrategia voraz mientras que la otra está guiada por un comportamiento aleatorio. Estas estrategias se aplican alternativamente y de forma iterativa, pudiendo ser complementada en cualquier caso por una búsqueda local.

Cualquier algoritmo *IG* tiene dos fases bien definidas:

- *Construcción:* esta fase se diseña en base a un algoritmo iterativo en el que, partiendo de una solución inicial (que puede ser la solución vacía o la solución total, dependiendo del

punto de partida) y, tras un número de pasos, se va conformando la solución final. Dicho proceso es voraz, porque en cada paso intenta incluir en la solución aquella opción que mejor resultado dé (que más haga aumentar el valor del *fitness*). Está fase finalizaría cuando fuese imposible encontrar un candidato cuya inclusión en la solución constructiva, repercutiera en una mejora del valor de la función *fitness*.

- *Aleatoria:* en esta fase se destruye parte de la solución obtenida en el paso anterior. Para añadir aleatoriedad a la solución, ampliando el horizonte de búsqueda de soluciones, se eliminan al azar y de la solución anterior, varias de las instancias seleccionadas en dicha fase. Esta cantidad de datos eliminados constituirá uno de los parámetros a ajustar en el algoritmo final.

Estos dos pasos son repetidos iterativamente hasta que se cumple una cierta condición de parada. Por lo tanto, y dentro del marco de una metaheurística *IG*, diseñamos nuestro algoritmo.

Lo primero que podemos destacar es la existencia de dos estrategias completamente opuestas para obtener la solución constructiva. En el primero de los casos podemos partir de las solución vacía (instancias sin atributos) e ir añadiendo columnas a estas instancias en cada paso. En cada una de las iteraciones se añadiría aquella característica cuya inserción en la tabla solución repercutiese en un mayor aumento de la función *fitness* (comportamiento completamente voraz), ya que de un lado se va empeorando el valor de la reducción pero se va mejorando el valor de la clasificación. Una vez finalizada esta parte y para la fase aleatoria, se eliminaría un tanto por cierto de las columnas añadidas en la fase anterior. A continuación se añadirían de nuevo columnas a la tabla reducida (siguiendo las pautas marcadas en la fase constructiva) y se eliminarían en base a la fase aleatoria, repitiendo este esquema hasta que se cumpliese la condición de parada. A esta primera implementación la denominaremos *IG-cd*.

En la segunda de las estrategias podemos partir de la tabla original completa (las instancias están representadas con todos sus atributos) e ir eliminando columnas en cada una de las iteraciones y siguiendo un procedimiento voraz (siempre que fuese posible mejorar la función *fitness*). En este caso, en cada iteración se puede ir empeorando (no tiene por qué) el porcentaje de clasificación, aunque a la vez se va mejorando el de reducción (al ir disminuyendo la cantidad de datos). Una vez finiquitada esta fase se añadirían aleatoriamente algunas de las columnas anteriormente rechazadas.

Al igual que ocurría en el planteamiento presentado con anterioridad, el proceso se repetiría hasta que se alcanzase la condición de parada. A esta segunda alternativa la representaremos con el acrónimo *IG-dc*.

Para ambos casos dicha condición fue fijada cuando, como resultado de la fase aleatoria, fuese imposible encontrar un candidato para la siguiente fase constructiva que mejorara el *fitness* obtenido hasta ese momento. Otra condición de parada es limitar el número de construcciones y mejoras a 100 iteraciones.

5. Resultados Experimentales

Para la realización de este trabajo, se emplearon una serie de conjuntos tomados del repositorio de la UCI, disponible en la siguiente dirección: http://archive.ics.uci.edu/ml/. Hemos elegido conjuntos conformados por un número de instancias inferior a 1000 elementos y con atributos de carácter numérico. Hemos desestimado el trabajar con bases de datos grandes, siendo esta una de las tareas a realizar en futuros trabajos. En la Tabla 2 se muestran las características de los mismos (a saber: nombre, identificador, número de instancias, número de columnas y número de clases).

Tabla 2. Conjuntos de datos utilizados en este trabajo.

Nombre	*Id*	*Inst.*	*Col*	*Cl*
SoyBean	Sb	307	35	19
Ionosphere	Io	351	34	2
Cleveland	Cv	432	6	2
Pima	Pm	768	8	2
Wisconsin	Wc	683	9	2
Vowel	Vw	528	10	11

Estas bases de datos se refieren a temas tan dispares como *Wisconsin*, donde los datos tomados a los pacientes de un hospital valen para clasificarlos entre personas enfermas y sanas. Por otro lado, la base de datos *SoyBeans* muestra tipos de enfermedades sufridas por la planta de soja.

Otra de las bases de datos trabajadas (*Ionosphere*) clasifica entre correctos y deficientes los datos recibidos por una serie de radares que analizan la ionosfera terrestre.

Siguiendo una estrategia de validación cruzada, cada uno de los conjuntos fue dividido en 5 partes que mantenían la relación de clases del conjunto inicial (5-*fold cross validation*). De esta forma cada partición permitía entrenar con el 80% de las instancias e ir testando la calidad de la tabla reducida (ya con un número más reducido de atributos) con el 20% de las instancias no utilizadas en la fase de entrenamiento. Sólo de esta manera es posible calibrar la bondad de la tabla reducida ya que, al contener ésta todas las instancias iniciales (aunque con menos atributos), el clasificador del vecino más cercano otorgaría a cada una de las instancias su propia clase (es la instancia más cercana que tiene). El procedimiento anteriormente presentado fue desarrollado en lenguaje Java, llevándose a cabo las ejecuciones en un ordenador Pentium IV Core 2 Duo 2.27 GHz y 2 GB de memoria RAM.

Dado que tenemos dos metodologías diferentes para aplicar sobre los mismos datos, empezaremos por la que tiene como base constructiva la de añadir atributos a la tabla vacía y cuya fase aleatoria elimina algunas de las columnas que conforman en ese momento la tabla reducida (*IG-cd*). Tras una serie de pruebas iniciales y para este caso en concreto, este porcentaje fue fijado en un 30%.

En la Tabla 3 aparecen los resultados medios para el porcentaje de clasificación de los diferentes conjuntos junto con el tiempo medio de ejecución por partición. La última fila de la tabla contiene los valores medios de todos los conjunto.

Tabla 3. Tiempos de ejecución y porcentajes de acierto para todos los conjuntos, aplicando el algoritmo *IG-cd*.

Id	*T (s)*	*Acierto (%)*
Sb	235	76.0 ± 9.8
Io	36.7	80.8 ± 2.0
Cv	2.29	43.8 ± 1.9
Pm	6.58	63.4 ± 3.6
Wc	5.28	90.4 ± 1.7
Vw	51.6	94.2 ± 1.7
Media	----	*74.8 ±17.1*

El rango de aciertos, para todos los conjuntos, está comprendido entre un 44% y un 94%. Es decir, que nos encontramos con bases de datos cuya función como clasificador es muy buena, mientras que otras bases de datos son difíciles de reproducir.

En la Tabla 4 se muestran los resultados obtenidos para el número de columnas seleccionadas por el algoritmo y su reflejo en el porcentaje de reducción.

Tabla 4. Número final de columnas y porcentajes de reducción para todos los conjuntos, aplicando el algoritmo *IG-cd.*

Id	*Col. In.*	*Col. Fin.*	*Reducción (%)*
Sb	35	11.8 ±0.4	66.3 ± 1.1
Io	34	4.8 ± 1.0	85.9 ± 2.9
Cv	6	3.4 ± 0.5	73.8 ± 3.8
Pm	8	2.6 ± 0.8	67.5 ± 10.0
Wc	9	3.2 ± 0.4	64.4 ± 4.4
Vw	10	4.2 ± 0.4	58.0 ± 4.0
Media	----	----	*69.3 ±2.8*

Destaca el nivel de reducción elevado que produce el algoritmo en la mayoría de las bases de datos, donde conjuntos de cientos de instancias son representados por una tabla reducida con instancias representadas por menos de la mitad de los atributos que tenían inicialmente.

En una segunda fase del trabajo, se implementó el algoritmo inverso al expuesto anteriormente (*IG-dc*). Es decir, que partimos de una tabla reducida donde todas las instancias contienen al 100% de sus atributos, los cuales van siendo descartados en las sucesivas iteraciones de la fase constructiva. Una vez finalizada dicha fase, se añaden aleatoriamente un tanto por ciento de las columnas descartadas en el paso anterior y se volverá a retomar la etapa constructiva, repitiéndose hasta que se cumpla la condición de parada. En este caso, y también en base a una serie de pruebas realizadas anteriormente, este porcentaje se fijó en un 30%.

Al igual que hicimos con la versión anterior del algoritmo voraz iterativo, en la Tabla 5 aparecen los resultados medios para el porcentaje de clasificación y los tiempos medios de ejecución por partición. También devolvemos, en la última fila de la tabla, el valor medio para el porcentaje de acierto de los conjuntos.

Tabla 5. Tiempos de ejecución y porcentajes de acierto para todos los conjuntos, aplicando el algoritmo *IG-dc*.

Id	T (s)	Acierto (%)
Sb	158	65.2 ± 6.3
Io	66.6	80.4 ± 3.2
Cv	9.5	30.8 ± 5.9
Pm	53.5	62.6 ± 6.6
Wc	29.2	90.2 ± 3.5
Vw	100	93.4 ± 1.9
Media	----	*70.4 ±21.1*

El rango de aciertos, para todos los conjuntos, está comprendido entre un 31% y un 93%. Sin embargo, los porcentajes, en su conjunto, son similares a los obtenidos por el algoritmo anterior.

En la Tabla 6 se muestran los resultados obtenidos para el número de columnas seleccionadas por el algoritmo y los porcentajes de reducción derivados de los mismos.

Tabla 6. Número final de columnas y porcentajes de reducción para todos los conjuntos, aplicando el algoritmo *IG-dc*.

Id	*Col. In.*	*Col. Fin.*	*Reducción (%)*
Sb	35	8.6 ±1.4	75.4 ± 4.0
Io	34	4.0 ± 0.9	88.2 ± 2.6
Cv	6	1.6 ± 0.5	87.7 ± 3.8
Pm	8	2.0 ± 0.1	75.0 ± 1.3
Wc	9	2.6 ± 0.5	71.1 ± 5.5
Vw	10	3.8 ± 0.7	62.0 ± 7.0
Media	----	----	*76.6 ±9.2*

De nuevo, los porcentajes de reducción son similares a los obtenidos por el algoritmo anterior.

Los resultados así expuestos no dan una visión completa de la bondad del método hasta que no son comparados con resultados pertenecientes al estado del arte. Es por ello que hemos comparados los mismos con los publicados en el trabajo de García y colaboradores [4], cuyo algoritmo de *Scatter Search* paralelo fue aplicado, entre otros, sobre estos conjuntos de datos.

Los resultados correspondientes a las dos versiones del algoritmo voraz iterativo son mostrados en la Tabla 7, juntos con los del trabajo mencionado con anterioridad. Hemos añadido una última fila a la tabla en la cual se encuentran los valores medios del *fitness.* En negrita aparecen resaltados los mejores resultados.

Tabla 7. Comparativa para los valores del *fitness* entre los algoritmos desarrollados en este trabajo y el presentado en el trabajo de García y colaboradores (identificado como SS). En negrita aparecen los mejores resultados.

Id	*IG-cd*	*IG-dc*	*SS*
Sb	71.2 ± 9.9	70.3 ± 7.5	**72.0± 2.8**
Io	83.4 ± 3.5	84.3 ± 4.1	**91.8 ± 1.4**
Cv	58.8 ± 4.2	59.3 ± 7.0	**66.1 ± 6.3**
Pm	65.5 ± 9.9	**68.8 ± 6.7**	57.8± 7.2
Wc	77.4 ± 4.7	**80.7 ± 6.5**	63.7 ± 9.5
Vw	76.1 ± 4.3	**77.7 ± 7.3**	56.8± 2.7
Media	*72.1± 8.1*	***73.5 ± 8.4***	*68.0 ± 12.9*

En valor medio, el algoritmo voraz iterativo (en sus dos versiones) supera al desarrollado por García y colaboradores, aunque habría que confirmar por medio de algún test estadístico y de manera más exacta, este resultado. No obstante podemos asegurar que los resultados obtenidos por el algoritmo *IG* de este trabajo se encuentran a la altura de los existentes en el estado del arte.

6. Conclusiones y trabajos futuros

En este trabajo se ha desarrollado un algoritmo voraz iterativo (con dos versiones) con objeto de aplicarlo al preprocesamiento de bases de datos. Concretamente su aplicación ha ido orientada hacia el problema de la selección de características representativas del conjunto de datos. El algoritmo diseñado se ha aplicado sobre varias bases de datos y los resultados se han comparados con los obtenidos por otro algoritmo. La comparativa ha mostrado que los resultados obtenidos por nuestro algoritmo reproducen los disponibles en el estado del arte.

Como futuros trabajos se encuentran el trabajar con conjuntos de mayor tamaño y el combinar dicho algoritmo con alguno de los desarrollados por el grupo para la reducción de instancias con objeto de desarrollar un procedimiento capaz de reducir el tamaño de una base de datos de manera bidimensional.

Referencias

[1] A. Araúzo, J.M. Benítez, J.L. Castro, Consistency measures for feature selection.

Journal of Intelligent Information Systems 30, 2008.

[2] J. Derrac, S. García, F. Herrera, IFS-CoCo: Instance and Feature Selection based on Cooperative Coevolution with Nearest Neighbor Rule. Pattern Recognition 43:6 2082-2105, 2010.

[3] Alex A. Freitas, Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer, 2002.

[4] Félix García López, Miguel García Torres, Belén Melián Batista, José A. Moreno Pérez, Marcos Moreno-Vega, "Solving Feature Selection Problem by a Parallel Scatter Search" European Journal of Operational Research, Volume 169, Issue 2, 477 - 489, 2006.

[5] Jiawei Han and Micheline Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, 2006.

[6] F. Herrera, J.R. Cano, Técnicas de reducción de datos en KDD: El uso de Algoritmos Evolutivos para la Selección de Instancias. Actas del I Seminario Sobre Sistemas Inteligentes (SSI'06), Universidad Rey Juan Carlos, Madrid, 165-181, 2006.

[7] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, 177(3), 2033–2049, 2007.

[8] R. Ruiz and T. Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research, 187(3), 1143–1159, 2008.