

# Nuevas heurísticas y medidas de textura para resolver problemas de *scheduling* mediante *clausura* y *CSP*

María Isabel Alfonso Galipienso

Dto. de Ciencia de la Comp. e IA. U. de Alicante. Ap. Correos, 99  
E-03080 Alicante, España. e-mail: [eli@dccia.ua.es](mailto:eli@dccia.ua.es)

Federico Barber Sanchís

Dto. de Sist. Informáticos y Comp. U. P. de Valencia. Cno. de Vera s/n  
E-46020 Valencia, España. e-mail: [fbarber@dsic.upv.es](mailto:fbarber@dsic.upv.es)

Palabras clave: heurísticas, *scheduling*, satisfacción de restricciones, *clausura* de restricciones, razonamiento temporal.

*Resumen. Un problema de scheduling puede plantearse como un problema de satisfacción de restricciones (CSP). En los procesos CSP, el uso de heurísticas de ordenación de variables y valores, constituyen elementos fundamentales a la hora de realizar una búsqueda eficiente de una solución. También puede aplicarse un proceso de clausura (previo a la aplicación del CSP) para reducir el espacio de búsqueda resultante. Recientemente se ha propuesto un nuevo método que integra de forma efectiva estos dos procesos (clausura y CSP) que tradicionalmente se han aplicado por separado. En este trabajo se presentan nuevas medidas de textura y heurísticas, así como la adaptación de heurísticas ya existentes, para su aplicación en el nuevo método. Así se consigue que el proceso sea empíricamente más eficiente, puesto que se pueden tomar mejores decisiones, obteniendo soluciones óptimas o muy cercanas al óptimo, como constatan los experimentos realizados, que también se exponen.*

## 1 Introducción

Un problema de *scheduling* [Baker74] puede caracterizarse como un conjunto de restricciones temporales, y por lo tanto, las técnicas desarrolladas en el marco de los problemas de Satisfacción de restricciones (*CSP*- Constraint Satisfaction Problems), y más concretamente, sobre restricciones temporales, pueden aplicarse para resolver problemas de *scheduling*. Principalmente se utilizan dos procesos típicos: la *clausura*,

como técnica deductiva de nuevas restricciones derivadas [Dechter96], [Stergiou00], [Barber00], y técnicas de búsqueda *CSP*, guiadas por heurísticas, [Sadeh96], [Nuijten98], [Beck00].

Una propuesta reciente [Alfonso01] consiste en un nuevo modelo de resolución de restricciones temporales que integra ambas técnicas (*clausura* y *CSP*) de forma efectiva, intentando aprovechar las ventajas de cada una de ellas. Adicionalmente, el modelo propuesto permite la definición y aplicación de nuevas heurísticas que, por una parte, permiten limitar considerablemente el número de disyunciones de la red y, por otra, permiten dirigir más eficientemente el proceso de búsqueda. Este trabajo se centra en el estudio y definición de las nuevas heurísticas que pueden utilizarse en dicho proceso integrado. Hay que destacar que, adicionalmente al problema de *scheduling*, el proceso propuesto puede aplicarse a problemas generales de satisfacción de restricciones. Se consideran tres tipos de heurísticas:

- Heurísticas ya existentes, pero que se han adaptado para contemplar el caso en el que se deba tratar con restricciones disyuntivas etiquetadas.
- Heurísticas y medidas de textura nuevas, que aprovechan la información global deducida por la propagación realizada por el proceso de *clausura*.
- Heurísticas combinadas, que permiten tomar mejores decisiones y proporcionan una mayor flexibilidad a la hora de definir diferentes estrategias de elección.

En primer lugar se hará una breve introducción al método de resolución que integra de forma efectiva las técnicas de *clausura* con *CSP*, centrándonos en la importancia de las heurísticas utilizadas para conseguir soluciones eficientes.

A continuación se introduce una nueva medida de textura para encontrar nuevas cotas superiores para la solución del problema de *scheduling*. En el siguiente apartado se indica cómo se adaptan algunas heurísticas existentes para contemplar el caso de restricciones disyuntivas. Se proponen dos heurísticas nuevas, y se explica el concepto de *heurísticas combinadas*. Finalmente se presentan algunos resultados y conclusiones finales.

## 2 Proceso integrado de *clausura* y *CSP*

El proceso integrado de *Clausura-CSP* propuesto en [Alfonso01] es un proceso iterativo en el que se alternan sucesivos procesos de *clausura* completa sobre restricciones métrico-disyuntivas [Barber00], con procesos *CSP-heurísticos*.

El modelo se caracteriza por aplicar un proceso de *clausura* completo sobre las restricciones y contempla dos parámetros básicos de entrada: (i) el número máximo de indecisiones (o disyunciones) que pueden mantenerse simultáneamente en el conjunto de restricciones actual, y (ii) las heurísticas que pueden aplicarse durante el proceso de

búsqueda. De esta forma, el proceso es capaz de compensar el elevado coste computacional que supone un proceso de *clausura* completo sobre restricciones disyuntivas, con el excesivo número de *backtrackings* que se suele producir en un proceso *CSP*, al efectuarse en cada momento el proceso de búsqueda sobre una única posible solución.

El uso de heurísticas adecuadas permite reducir el tiempo de cómputo guiando la búsqueda de forma que evitemos recorrer el máximo número posible de estados que no conducen a ninguna solución. En este caso, se ha aprovechado la mayor información propagada por el modelo de *clausura*, para introducir nuevas heurísticas, así como para adaptar el uso de heurísticas ya existentes, para que permitan considerar restricciones disyuntivas.

### 3 Obtención de nuevas cotas superiores

Cuando tratamos con problemas en los que no está limitado el tiempo de terminación de los trabajos (*deadline*), el tiempo mínimo para completar todos los trabajos (*makespan*), no está acotado superiormente, de forma que pueden existir infinitas soluciones. Cuantas más sean las soluciones posibles, el espacio de búsqueda es mayor, con lo que el coste computacional de encontrar una solución también aumenta.

Por lo tanto, es interesante la obtención de cotas superiores ajustadas del *makespan* para así limitar el tiempo de cómputo para encontrar una solución (o bien deducir que el problema no tiene solución).

Suponiendo que las disyunciones de la red tienen cardinalidad  $k$ , y considerando dos operaciones no secuenciadas  $(o_i, o_j)$ , que utilizan un mismo recurso  $r$  pertenecientes cada una a un trabajo distinto,  $o_i \in J_i$ ,  $o_j \in J_j$ ,  $J_i \neq J_j$ , damos las siguientes definiciones:

**Definición 1.** El máximo retraso que puede sufrir el trabajo  $J_j$  si las operaciones son secuenciadas en el orden  $o_i \rightarrow o_j$ , y que denominaremos  $\Delta D(J_j, o_i \rightarrow o_j)$ , vendrá dado por el valor máximo de los retrasos, para cada una de las  $k$  alternativas,  $\Delta_z(J_j, o_i \rightarrow o_j)$ , de la disyunción:

$$\Delta D(J_j, o_i \rightarrow o_j) = \max_{k=1..z} \Delta_z(\max(est_z(o_i) - est_z(o_j) + dur(o_i), 0))^1.$$

**Definición 2.** Una cota superior del *makespan*, si elegimos la secuencia de ordenación  $o_i \rightarrow o_j$ , y que denominaremos *TopMak*, vendrá dada por el valor máximo entre los tiempos más tempranos de finalización de los trabajos  $J_1, J_2, \dots, J_{j-1}, J_{j+1}, \dots, J_n$ , y el tiempo más temprano de finalización del trabajo  $J_j$ , teniendo en cuenta el máximo retraso que puede sufrir como consecuencia de dicha ordenación. Es decir:

---

<sup>1</sup> Denominamos  $est_z(o_i)$  al tiempo más temprano de comienzo teniendo en cuenta el intervalo  $z$ -ésimo de la restricción disyuntiva correspondiente.

$TopMak(o_i \rightarrow o_j) = \max(est_z(last(o_i)) + \Delta_z(J_j, o_i \rightarrow o_j), makespan(J_1, J_2, \dots, J_{j-1}, J_{j+1}, \dots, J_n))$ .

Esta nueva medida de textura la aplicaremos antes de seleccionar una nueva restricción disyuntiva para su procesamiento. Además, podemos obtener nuevas cotas superiores también cuando los *deadlines* de cada trabajo son conocidos, con objeto de reducir dichos valores, y así limitar el espacio de búsqueda.

## 4 Adaptación de heurísticas existentes

A partir de las heurísticas basadas en *slack* propuestas en [Smith93], éstas son adaptadas para el caso en el que haya disyunciones en la red. Tanto la heurística de ordenación de variables como la de valores propuesta tiene carácter **local**, es decir, utiliza información derivada de un análisis local de las variables (procesos de *clausura* parciales). Originalmente, dichas heurísticas *siempre* toman una única decisión. Esto puede ser un inconveniente si se quiere dotar de carácter *incremental*<sup>2</sup> a la solución buscada, necesario en el caso de que no conozcamos a priori la totalidad de las restricciones del problema, o éstas se vayan generando dinámicamente. La aportación realizada, consiste precisamente en la adaptación de dichas heurísticas para que se obtenga como resultado una única decisión (si así se desea), o un subconjunto de las decisiones posibles (o el conjunto total, que equivaldría a no tomar ninguna decisión). Esto nos va a permitir *retrasar* las decisiones de ordenación correspondientes hasta que obtengamos suficiente información (dejamos indecisiones pendientes).

En lo sucesivo, vamos a asumir que cada restricción de la red tiene una aridad igual a  $k$  (es decir, está formada por  $k$  intervalos temporales). En [Alfonso01] se definen los conceptos de  $slack_z(o_i, o_j)$ , así como  $slackD(o_i, o_j)$ , consistente este último al conjunto de formado por el valor de *slack* asociado a cada una de las  $k$  disyunciones.

Dado el conjunto de restricciones disyuntivas pendientes de secuenciar, equivalentes a pares de operaciones  $(o_p, o_q)$  que compiten por un mismo recurso, se definen las siguientes heurísticas:

**Definición 3.** Una nueva heurística de ordenación de variables consiste en elegir como siguiente par de operaciones a secuenciar, aquel cuyo valor de *slack* sea mínimo teniendo en cuenta todos los pares de operaciones sin secuenciar, es decir:

$$\min\{slack_z(o_i, o_j), slack_z(o_j, o_i)\} = \min_{p,q} \{\min\{slack_z(o_p, o_q), slack_z(o_q, o_p)\}, \\ \text{con } slack_z(o_p, o_q) \in slack_z(o_p, o_q), slack_z(o_q, o_p) \in slackD(o_q, o_p), \forall (o_p, o_q)\}.$$

**Definición 4.** La nueva heurística de selección de valores trata de decidir, dado un par de operaciones no ordenadas  $(o_i, o_j)$ , un secuenciamiento, de los dos posibles. Para

---

<sup>2</sup> Las variaciones o modificaciones de las restricciones de entrada pueden asumirse sin necesidad de rehacer todo el proceso

ello se escoge aquella decisión cuyo  $slackD$  temporal sea máximo (puesto que deja mayor grado de libertad en las siguientes etapas de la búsqueda). Es decir:

Si  $\max(slackD_z(o_i, o_j)) > \max(slackD_z(o_j, o_i))$ ,  $z=1..k$ , entonces elegimos  $o_i \rightarrow o_j$

Si  $\max(slackD_z(o_i, o_j)) < \max(slackD_z(o_j, o_i))$ ,  $z=1..k$ , entonces elegimos  $o_i \rightarrow o_j$ .

Si  $\max(slackD_z(o_i, o_j)) = \max(slackD_z(o_j, o_i))$ ,  $z=1..k$ , entonces no tomamos ninguna decisión.

Adicionalmente, se va a utilizar *edge-finding*, aplicado a intervalos de tareas [Ca-seau95], para obtener una heurística de ordenación de variables. La idea es elegir el par de operaciones sin secuenciar que pertenezcan a uno o más recursos considerados *críticos* [Alfonso01].

## 5 Heurísticas nuevas

Proponemos una nueva heurística que utiliza información local, basada en un nuevo concepto que hemos denominado "*lateness*". En un problema de scheduling, cada operación  $o_q$ , que se ejecuta sobre una máquina determinada, tiene una operación siguiente  $o_{q+1}$ , perteneciente al mismo trabajo, que se ejecuta sobre otro recurso distinto (excepto si se trata de la última operación de dicho trabajo). Por otro lado, esa misma operación  $o_q$  guardará una relación de orden "antes o después de" con  $o_p$ , perteneciente a otro trabajo. Un retraso en la operación  $o_q$  causará posiblemente un retraso en  $o_{q+1}$ , disminuyendo así la holgura total de  $o_{q+1}$ , y favoreciendo que sucesivas ordenaciones retrasen la terminación del trabajo al que pertenecen  $o_q$  y  $o_{q+1}$ .

Vamos a definir una nueva heurística para decidir el siguiente par de operaciones a secuenciar, basada en considerar cómo afecta una cierta decisión de ordenación de dos operaciones  $o_p \rightarrow o_q$ , que compiten por un mismo recurso, sobre la operación siguiente a la segunda de ellas,  $o_{q+1}$  perteneciente al mismo trabajo.

Supongamos que las restricciones de la red tienen aridad  $k$ . Dado un par de operaciones pendientes de secuenciar  $o_p$  y  $o_q$ , que compiten por un mismo recurso, y una posible ordenación  $o_p \rightarrow o_q$ , realizamos las siguientes definiciones:

**Definición 5.** Denominamos  $est\_nextD(o_p, o_q)$  a los  $k$  tiempos de comienzo más temprano de la operación  $o_{q+1}$ , sucesora de  $o_q$ , antes de realizar el secuenciamiento  $o_p \rightarrow o_q$ , y cuyo valor viene dado por la expresión:

$$est\_nextD_z(o_p, o_q) = est_z(o_q) + dur_z(o_q), z = 1..k.$$

**Definición 6.** Denominamos  $new\_est\_nextD(o_p, o_q)$  a los  $k$  tiempos de comienzo más temprano de la operación  $o_{q+1}$ , sucesora de  $o_q$ , después de realizar el secuenciamiento  $o_p \rightarrow o_q$ , y cuyo valor viene dado por la expresión:

$$new\_est\_nextD(o_p, o_q) = \max(estD_z(o_p) + dur_z(o_q) + dur_z(o_p), estD_z(o_q) + dur_z(o_q))$$

**Definición 7.** Denominaremos *lateness\_nextD*( $o_p, o_q$ ) a las  $k$  diferencias entre los  $k$  tiempos más temprano de comienzo de la operación  $o_{q+1}$  como resultado de la ordenación, y los  $k$  tiempos más temprano de comienzo de dicha operación antes de la ordenación, correspondientes a cada una de las  $k$  disyunciones de las restricciones. Así:

$$lateness\_nextD_z(o_p, o_q) = \{new\_est\_nextD_z(o_p, o_q) - est\_nextD_z(o_p, o_q)\}_{z=1..k}$$

**Definición 8.** La nueva heurística de ordenación de variables consiste en elegir como siguiente par de operaciones a secuenciar, aquél cuyo valor de *lateness\_nextD* sea mínimo, teniendo en cuenta todos los pares de operaciones pendientes de secuenciar. Es decir:

$$\min\{lateness\_nextD_z(o_p, o_q), lateness\_nextD_z(o_q, o_p)\} = \{\min_{(p,q)}\{lateness\_nextD_z(o_p, o_q), lateness\_nextD_z(o_q, o_p)\}\} \forall (o_p, o_q) \text{ sin secuenciar.}$$

Si hay varios mínimos, la heurística devolverá como resultado uno de los pares de operaciones asociado a dicho valor mínimo.

**Definición 9.** La nueva heurística de ordenación de valores consiste en, dadas dos operaciones no ordenadas  $o_p$  y  $o_q$ , se calcula el valor de *lateness\_nextDmin* para cada ordenación de las dos posibles, y se elige la menor de los dos (con ello estamos penalizando las ordenaciones que provocan un mayor aumento del *makespan*):

Si  $\min(lateness\_nextD_z(o_p, o_q)) < \min(lateness\_nextD_z(o_q, o_p)), z=1..k$ , elegimos  $o_p \rightarrow o_q$

Si  $\min(lateness\_nextD_z(o_p, o_q)) > \min(lateness\_nextD_z(o_q, o_p)), z=1..k$ , elegimos  $o_q \rightarrow o_p$

Si  $\min(lateness\_nextD_z(o_p, o_q)) = \min(lateness\_nextD_z(o_q, o_p)), z=1..k$ , no tomamos ninguna decisión.

Otra heurística nueva con carácter global (utiliza información deducida por el proceso de clausura total) es la que se obtiene utilizando el valor del *makespan* actual [Alfonso01].

## 6 Concepto de heurísticas combinadas

Debido al hecho de que permitimos mantener disyunciones en la red de restricciones, las heurísticas utilizadas pueden no tomar ninguna decisión, o incluso tomar múltiples elecciones en un mismo momento. Para obtener mejores resultados, podemos aplicar varias heurísticas para "reforzar" o "rechazar" la decisión tomada por alguna de ellas. La idea es aprovechar la información que proporciona el método de *clausura* utilizado,

para tomar mejores decisiones y evitar backtrackings, permitiendo soluciones parciales obtenidas más flexibles. De esta forma, podemos combinar el resultado de varias heurísticas, creando así otras nuevas que puedan adaptarse a las características particulares del problema, o de la solución que queramos obtener. Además, como el uso de heurísticas es parametrizable, podemos cambiar los criterios de elección sin cambiar de método de resolución.

En [Alfonso01] se definen dos heurísticas combinadas, una de ordenación de variables, denominada *makespan\_slack*, y otra de ordenación de valores *edge\_slack*. Se pueden utilizar nuevas heurísticas teniendo en cuenta las nuevas heurísticas basadas en *lateness*. Así tendríamos la combinación *makespan\_lateness*, o bien *lateness\_makespan*, que pueden utilizarse ambas tanto para ordenación de variables como de valores. En cuanto al criterio de decisión utilizado, se puede considerar que si una de las dos heurísticas no decide, pero la otra sí, entonces consideramos válida la decisión de una de ellos. Si las dos deciden, se toma la intersección de ambas. Si ningún criterio es capaz de tomar una decisión, entonces se mantendría la indecisión.

## 7 Evaluación

En primer lugar vamos a ver el efecto del uso de distintas heurísticas con respecto al número de *backtrackings* realizados. Se han generado un conjunto de 30 muestras aleatorias *job-shop*  $10 \times 5$  basándonos el modelo utilizado en [Sadeh96]. Se trata de muestras estándar en la bibliografía, de un tamaño (complejidad) considerable y que incluyen cuellos de botella a resolver. En la Figura 1 se muestra una gráfica en la que se ha considerado un recurso *cuello de botella*, y un valor de ajuste inicial del *makespan* igual a 0 (RG), que posteriormente se ajusta en un 10% inferior a dicho valor.

Se han utilizado 3 combinaciones distintas de heurísticas: en la primera de ellas (*edge+slack*) se ha usado la heurística de selección de variables basada en *edge-finding*, juntamente con la heurística de selección de valores basada en *slack*; la segunda (*slack+makespan*) utiliza la heurística basada en *slack* para seleccionar variables y la heurística que tiene en cuenta el *makespan*, para seleccionar valores; finalmente **Heur.Combin** utiliza la heurística combinada de ordenación de variables *edge\_slack*, y la heurística combinada *makespan\_slack* para la selección de valores. En cualquier caso se ha utilizado adicionalmente la heurística para limitar el final del *scheduling TopMak* en sucesivas adiciones de restricciones disyuntivas.

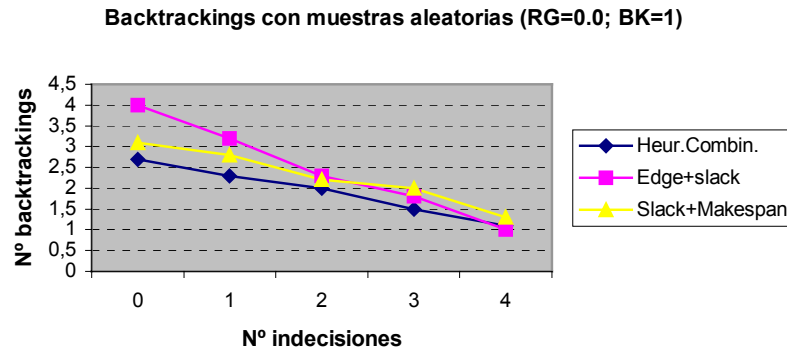


Figura 1. Efecto de distintas combinaciones de heurísticas.

En el eje de ordenadas se representan el número de backtrackings realizados hasta encontrar una solución. Se puede observar que dicho número se ve condicionado por el número de indecisiones mantenidas en la red de restricciones y por las heurísticas utilizadas. A medida que el número de indecisiones aumenta, y dependiendo de las heurísticas utilizadas se requieren menos procesos de *backtracking*. Podemos observar en este caso, que con el uso de una heurística combinada de ordenación de valores se obtiene una mayor reducción del número de *backtrackings* necesarios que con el resto. Naturalmente, el permitir indecisiones conlleva un incremento del coste temporal del algoritmo, pero por contra, se obtienen mejores soluciones (más cercanas al óptimo). Para las muestras de la Figura 1 se obtienen unos tiempos de alrededor de 5 segundos (para procesar las 30 muestras) cuando se permiten 4 indecisiones, frente a 1 segundo si el número de indecisiones mantenidas es cero. Además, el tiempo de cómputo de las heurísticas es bajo (despreciable frente al coste de la búsqueda). En cualquier caso, la implementación disponible del método es todavía un prototipo, utilizando lenguaje Lisp, (con fuentes compilados, bajo linux Redhat 5.1), ejecutándose sobre un procesador Pentium II con 128Mb.

Para el segundo experimento, que se muestra en la Figura 2, se han utilizado muestras de un conocido banco de pruebas<sup>3</sup> (*benchmark*), con distintos tamaños. En la tabla de la figura se muestra la solución obtenida utilizando distintas combinaciones de heurísticas de selección de valores y variables, que hemos denominado **C1**,...,**C4**.. Concretamente **C1** corresponde al uso de las heurísticas de selección de variables y valores basadas en el cálculo de *lateness\_next*, **C2** corresponde al uso de la heurística de selección de variables basada en *edge-finding* y la heurística de selección de valores combinada *makespan\_slack*, **C3** usa la heurística de selección de variables *slackDmin*, y la heurística de selección de valores basada en el *makespan*, y **C4** combina la heurística

<sup>3</sup> Se puede obtener en la dirección: <ftp://ftp.mscmga.ms.ic.ac.uk/pub/>.

combinada de selección de variables *edge\_slack* y la heurística de selección de valores basada en el cálculo de *lateness\_next*. En la última columna se muestra la solución óptima para cada una de las instancias.

instancia	C1	C2	C3	C4	sol. ópt.
<i>car1</i>	7038	7038	7044	7038	7038
<i>car2</i>	7182	7166	7170	7173	7166
<i>car3</i>	7313	7312	7312	7312	7312
<i>car4</i>	8043	8003	8024	8035	8003
<i>car5</i>	7702	7702	7742	7742	7702
<i>ft06</i>	55	55	55	55	55
<i>ft10</i>	932	930	932	937	930
<i>la01</i>	668	666	666	666	666
<i>la02</i>	659	655	655	675	655
<i>la06</i>	965	926	926	955	926
<i>la16</i>	956	947	949	953	945
<i>la21</i>	1055	1046	1055	1048	1046

Figura 2. Efecto de distintas combinaciones de heurísticas.

Según se aprecia en la tabla, la columna C2 es la que proporciona los mejores resultados, debido al uso de heurísticas combinadas, que permiten tomar mejores decisiones. Del resto, C3 y C4 proporcionan, en la mayoría de casos, mejores resultados que C1. Ello es debido a que las heurísticas de C1 únicamente utilizan información local, mientras que C3 contempla una heurísticas que maneja información global, y C4 utiliza una heurística combinada.

## 8 Conclusiones

En este trabajo se han propuesto nuevas heurísticas y medidas de textura, y se han adaptado heurísticas existentes, que han sido aplicadas con éxito a un nuevo método de resolución de problemas de restricciones temporales [Alfonso01]. Para ello se han mostrado algunos de los resultados obtenidos utilizando tanto instancias generadas aleatoriamente, como instancias de bancos de pruebas conocidos. Dichos resultados, ponen de manifiesto que con el uso de dichas heurísticas se consigue dotar de eficiencia a dicho método, consiguiendo reducir el número de *backtrackings* realizados, y obteniendo soluciones más cercanas al óptimo en todos los casos, consiguiendo soluciones óptimas en la mayoría de ellos.

Como trabajos futuros proponemos la aplicación de dichas heurísticas a otros métodos, así como investigar el uso de nuevas combinaciones de heurísticas, con distintos criterios de decisión.

## Referencias

- [Alfonso01] Alfonso, M.I., and Barber, F. “*A mixed closure-CSP meted to solve scheduling problems*”. Engineering of intelligent systems. 14<sup>th</sup> Int. Conf. on Ind. and Eng. Applic. of A.I. and Expert Systems (IEA/AIE-2001). Springer-Verlag. LNAI 2070 (2001) 559-570
- [Baker94] Baker, A. “*The hazards of fancy backtracking*”. Proceedings of the Twelfth National Conference on Artificial Intelligence. Menlo Park, Calif.: American Association for Artificial Intelligence (1994) 288-293
- [Barber00] Barber, F. “*Reasoning on interval and point- based disjunctive metric constraints in temporal contexts*”. JAIR 12 (2000) 35-86
- [Beck00] Beck, J.C., Fox M.S. “*Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics*”. Art. Intel., 117 (2000) 31- 81.
- [Caseau94] Caseau Y. and Laburthe, F. “*Disjunctive scheduling with task intervals*” ICLP’94 Proc. of the 11 Int. Conf. on Logic Prog., MIT Press (1994)
- [Dechter96] Dechter, R. and van Beeki, P. “*Local an global relational consistency*”. Technical Report. University of California, Irvine (1996)
- [Nuijten98] Nuijten, W.P.M. and Le Pape, C. “*Constraint-based job shop scheduling wiht ILOG scheduler*”. Journal of heuristics, March, 3 4 (1998) 271-286.
- [Sadeh96] Sadeh, N. and Mark, S. Fox. “*Variable and value ordering heuristics for the job- shop scheduling constraint satisfaction problem*”. Artificial Intelligence 86 1 (1996) 1-41
- [Smith93] Smith, S.F., and Cheng, C.C. “*Slack-based heuristics for constraint satisfaction scheduling*”. Proceed. of the 11<sup>th</sup> Nat. Conf. on A.I., Menlo Park, Calif.: American Association for A. I. (1993) 139-144
- [Stergiou00] Stergiou, K. and Koubarakis, M. “*Backtracking algorithms for disjunctions of temporal constraints*”. Artif. Intellig. 120 (2000) 81-117