Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial
ISSN: 1137-3601
revista@aepia.org
Asociación Española para la Inteligencia Artificial
España

Alfonso Galipienso, María Isabel; Barber, Federico
Combinación de procesos de clausura y CSP para la resolución de problemas de scheduling
Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, vol. 4, núm. 9, invierno, 2000, pp. 20 - 26
Asociación Española para la Inteligencia Artificial
Valencia, España

Combinación de procesos de clausura y CSP para la resolución de problemas de scheduling

María Isabel Alfonso Galipienso y Federico Barber

Dept. de Ciencia de la Computación e Inteligencia Artificial. Univ. de Alicante, Alicante
Dept. de Sistemas Informáticos y Computación. Univ. Politécnica de Valencia, Valencia
eli@dccia.ua.es,fbarber@dsic.upv.es

ABSTRACT. *El problema de scheduling ha sido estudiado bajo diferentes aproximaciones, fundamentalmente mediante técnicas CSP. En este artículo se presenta un método que combina el proceso de clausura de restricciones con el proceso CSP. Inicialmente, modelamos el scheduling como el problema de satisfacer y encontrar la solución de un conjunto de restricciones métricas disyuntivas, basadas en puntos de tiempo. El método se basa en la adición sucesiva de restricciones (constraint-posting), efectuando un proceso de clausura total en cada nueva adición. Además, para limitar la complejidad del problema, se aplica un proceso CSP parcial que limita el conjunto de posibles soluciones, sin llegar a una instanciación de las variables. Los criterios de decisión están basados en heurísticas locales y globales que permiten mantener un conjunto limitado de soluciones y reducir el número de backtrackings necesarios para llegar a una solución óptima. Una vez procesado el conjunto de restricciones, se obtiene el conjunto mínimo de soluciones, de entre las que podemos obtener una cualquiera de ellas mediante sucesivas instanciaciones de las variables, ya sin necesidad de hacer backtracking.*

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

1.- INTRODUCCION.

El problema de scheduling es un problema de complejidad NP-completo (Garey79) y es tradicionalmente resuelto mediante técnicas CSP *(Constraint Satisfaction Problem)* (Mackworth77), que intentan reducir el espacio de búsqueda, y así limitar la complejidad experimental del problema. En una aproximación *CSP*, resulta fundamental la elección de heurísticas para una ordenación de instanciación de variables y valores en sus respectivos dominios (Sadeh96). El objetivo de estas heurísticas es limitar la exploración del árbol de soluciones, así como el número de bactrackings necesarios para obtener una solución (Beck98).

Por otra parte, el problema de scheduling puede ser formalizado como un problema de satisfacción de restricciones temporales (LePape88), (Belhadji98). En la mayoría de casos, estas restricciones son métrico-disyuntivas basadas en puntos de tiempo. Los procesos básicos sobre restricciones temporales son los procesos de consistencia (que permiten asegurar que existe solución) y de clausura (que permite obtener las restricciones mínimas) (Dechter91).

En este artículo proponemos un nuevo método para la resolución de problemas de scheduling. Este método intenta combinar un proceso de clausura de restricciones con un proceso CSP. Además, se amplía la base de las heurísticas a aplicar, combinando el uso de heurísticas locales, provenientes de la técnica CSP, con heurísticas globales, derivadas de la información que nos

proporciona el proceso de clausura (restricciones mínimas). El resultado será la obtención de una solución que minimice el tiempo necesario para completar la ejecución de todas las tareas del problema y reduciéndose la necesidad de hacer backtracking durante el proceso de búsqueda de dicha solución. El proceso de clausura implica un incremento de la complejidad temporal y espacial del algoritmo. Trataremos de mostrar que el método se puede flexibilizar para que este incremento de complejidad sea asumible, de forma que obtengamos una solución en un tiempo razonable. Por ello, la propuesta se enmarca en una alternativa de resolución de problemas de scheduling integrando técnicas de clausura y CSP.

El artículo se organiza de la siguiente forma: el apartado 2 definimos el problema de scheduling y cómo puede plantearse como un problema de Satisfacción de Restricciones Temporales (*Temporal Constraint Satisfaction Problem -TCSP)* (Dechter91). A continuación describimos brevemente la aproximación *CSP*, como proceso búsqueda de una solución al problema. En el apartado 4 proponemos nuestra alternativa, que combina las dos aproximaciones anteriores, manteniendo un conjunto de soluciones a la vez, mediante clausura. Finalmente analizamos el comportamiento del algoritmo con algunos ejemplos concretos y se presentan las conclusiones y trabajos futuros.

2.- EL PROBLEMA DE SCHEDULING.

El problema de scheduling se define mediante un conjunto $\boldsymbol{J}$ de trabajos, y un conjunto $\boldsymbol{M}$ de recursos[1]: $\boldsymbol{J}=\{j_i\}/\ |\{j_i\}|=\boldsymbol{n}$, $\boldsymbol{M}=\{m_j\}\ /\ |\{m_j\}|=\boldsymbol{m}$. La producción de un trabajo determinado j_i, requiere la ejecución de una secuencia de operaciones $\{o_{ij}\}$, que denotan el uso del recurso m_j en el trabajo j_i. Llamamos $\boldsymbol{O}$ al conjunto de operaciones especificadas en el scheduling: $\boldsymbol{O}=\{o_{ij}\}/\ |\{o_{ij}\}|=\boldsymbol{n}\cdot\boldsymbol{m}^2$. Como restricciones tecnológicas, cada operación $o_{ij}\in\boldsymbol{O}$ requiere el uso exclusivo de un recurso $m_j\in\boldsymbol{M}$ durante un determinado tiempo de procesamiento $dur(o_{ij})$. Además, las operaciones de un mismo trabajo deben ejecutarse en un orden determinado, $precedencia(o_{ij},o_{(i+1)j})$, de forma que $o_{(i+1)j}$ no puede comenzar hasta que o_{ij} haya terminado completamente. Las operaciones que comparten una misma máquina son *no interrumpibles* y mutuamente exclusivas. El objetivo del proceso es encontrar una asignación de tiempos a las operaciones de forma que se completen todos los trabajos en el mínimo tiempo posible.

A la secuencia de ordenación de las operaciones la llamaremos *patrón de flujo.* Si todos los trabajos tienen la misma secuencia de ordenación de las operaciones, definimos el problema como *flow-shop*, mientras que si cada trabajo tiene una ordenación determinada lo denominamos *job-shop scheduling*. Otros patrones de flujo básicos son *permutation flow-shop*, en el que hay una ordenación determinada de las operaciones para cada uno de los trabajos, y también hay un orden de uso de cada uno de los recursos por parte de las operaciones; y *open-shop*, en el que no hay ninguna restricción de ordenación.

Los primeros trabajos sobre scheduling se iniciaron desde el campo de la Investigación Operativa (Johnson 54), resolviendo instancias flow-shop para dos máquinas. El problema se caracteriza por su reducción a una formulación matemática y se soluciona con métodos algorítmicos formales. Posteriormente, se han utilizado distintos métodos iterativos como *simulated anealing* (Zweben93), *tabu search* (Glover93), y algoritmos genéticos (Fox91), que pueden resolver el problema de una manera más general, incorporando y mejorando el uso de heurísticas. Se han desarrollado otras aproximaciones, desde la Inteligencia Artificial basadas en el conocimiento y satisfacción de restricciones, (Fox84), (Sadeh91), obteniéndose buenos resultados (Jain99), aunque el problema sigue abierto debido a su intratabilidad, con la meta de obtener procesos experimentalmente satisfactorios.

2.1. Restricciones temporales asociadas con un problema de scheduling.

Considerando el papel fundamental del tiempo en el problema que nos ocupa, vamos a establecer un marco temporal en el que identificaremos los trabajos y operaciones, así como las restricciones entre ellos. La primitiva temporal utilizada será el *punto de tiempo*, $t_i\ \hat{I}\ \boldsymbol{T}$. Sobre estos puntos de tiempo establecemos restricciones métrico disyuntivas (Dechter91), cuya forma general será: $t_i\ \{[d_1\ D_1],\ [d_2\ D_2],\ ....\ [d_n\ D_n]\}\ t_j$, con $d_i \le D_i$ representando:

$$(t_j - t_i \le [d_1\ D_1]) \vee (\boldsymbol{t_j} - \boldsymbol{t_i} \le [d_2\ D_2]) \vee ..... \vee (\boldsymbol{t_j} - \boldsymbol{t_i} \le [d_n\ D_n])$$

Así, la expresión $t_i\ \{[5\ 5],[10\ 20]\}\ t_j$, significa que "t_j está a 5 unidades de t_i o bien entre 10 y 20 unidades de tiempo de t_i". Es decir, $(t_j - t_j\ £\ [5,5]) Ú (t_j - t_j\ £\ [10,20])$

---

[1] Hablaremos indistintamente de recursos o máquinas.

[2] Asumimos que cada trabajo requiere el uso de todos los recursos. Adicionalmente, por simplicidad asumimos unas particularizaciones del problema general de scheduling, que pueden ser fácilmente extensibles.

Consideraremos un modelo del tiempo ***T*** discreto y finito. Definimos dos puntos especiales ***T0*** y ***Tf*** como tiempo inicial del scheduling (origen del mundo) y tiempo final (final del mundo) respectivamente.

Una **operación** $o_{ij} \in \boldsymbol{O}$ consistirá en un par de puntos temporales $on(o_{ij})=t^-_{ij}$ y $off(o_{ij})=t^+$, que cumplen que $on(o_{ij}) \pounds off(o_{ij})$, es decir: $on(o_{ij}) \{[0 ¥]\} off(o_{ij})$. La *duración* de una operación $dur(o_{ij})=d_{ij}$, podremos expresarla como $on(o_{ij}) [d_{ij} d_{ij}] off(o_{ij})$.

Un **trabajo** $j_i \in \boldsymbol{J}$, está fomado por una secuencia ordenada de operaciones $(o_{i0},o_{i1},\ldots,o_{im})$. Las restricciones de *precedencia* entre las operaciones tendrán la forma $"j=1..m, off(o_{ij})\{[0 ¥]\} on(o_{i(j+1)})$, que denominaremos $precedencias(J_i)$. Opcionalmente podemos representar un lapso de tiempo $[d_{min}\ D_{min}]$ a partir del cual puede comenzar un trabajo j_i (***ready time***): $\boldsymbol{T0}\ \{[d_{min}\ D_{min}]\}\ on(o_{i0})$, así como un intervalo de tiempo máximo de finalización del trabajo (***due time*** o *deadline*) $[d_{max}\ D_{max}]$: $\boldsymbol{T0}\ \{[d_{max}\ D_{max}]\}\ off(o_{im})$.

Cada operación debe ejecutarse sobre una **máquina** distinta, de forma exclusiva, dando lugar a las restricciones *disyuntivas*: $on(o_{uj})\ \{[dur(o_{uj}) ¥], [-¥\ dur(o_{vj})]\}\ on(o_{vj})$. Es de destacar que esta restricción impone una disyuntiva, lo cual determina la alta complejidad del proceso. La elección sobre estas disyuntivas será la base del método de búsqueda propuesto.

El **patrón de flujo** viene determinado por las restricciones de precedencia para cada uno de los trabajos: $\{precedencias(J_i)\},\ " i=1..n$. Si se trata de flow-shop scheduling, la ordenación de los elementos de cada trabajo será la misma: $precedencias(J_x)=precedencias(J_y)," x,y=1..n$. Y podrá ser distinta si estamos ante un job-shop. A la hora de realizar los experimentos, hemos elegido únicamente instancias flow-shop y job-shop. Pero el método podría también aplicarse a otros patrones, ya que solamente maneja las restricciones temporales derivadas ellos.

Utilizaremos una *Red de Restricciones Temporales* -TCSP- (Dechter 91) para representar todos los elementos del problema de scheduling[3]. Nuestro objetivo será obtener la secuencia de ordenación de cada una de las operaciones sobre los recursos para realizar todos los trabajos en el mínimo tiempo necesario (*makespan*).

[3] Se han efectuado algunas restricciones sobre el problema general de scheduling. Sin embargo, el planteamiento de la propuesta puede ser fácilmente extendido, adaptando la tipología de las restricciones. Concretamente, pueden aplicarse las propuestas en (Johnson98).

3.- APROXIMACIÓN CSP.

Un problema de satisfacción de restricciones *(CSP)* consiste en un conjunto de *variables* $\boldsymbol{Z}= \{x_1,\ldots x_n\}$, con dominios discretos y finitos $\boldsymbol{D}=\{D_1,\ldots D_n\}$, y un conjunto de ***m*** *restricciones* $\boldsymbol{C}=\{c_1,\ldots c_n\}$, que son predicados $c_k(x_i,\ldots x_j)$ definidos sobre el producto Cartesiano $D_i\times\ldots\times D_j$. Si $c_k(x_i=v_i,\ldots x_j=v_j)$ es cierto, se dice que la evaluación de las variables es *consistente* con respecto a c_k , o lo que es equivalente, c_k se *satisface*. Una *solución* es una asignación de un valor a cada variable, en su dominio respectivo, tal que todas las restricciones se satisfacen. Formulaciones más generales pueden encontrarse en (Freuder89) y (Mackworth85).

Una instancia de un *CSP* (***Z,D,C***), puede representarse como un grafo de restricciones (o red de restricciones) G={***V,E***}. Para cada variable *v* en ***Z***, hay un nodo *n* en ***V***. Para cada conjunto de variables conectadas por una restricción *c* en ***C***, hay un hiperarco *e* correspondiente de ***E***, (arco binario si las restricciones son binarias). El problema de scheduling puede verse como un caso especial del problema de satisfacción de restricciones, (Fox84).

Un problema CSP se resuelve básicamente instanciando sistemáticamente las variables y comprobando si satisfacen las restricciones. Ello da lugar a un coste computacional exponencial (Dechter91). Por ello, se suelen aplicar técnicas para decrementar el coste experimental, minimizando bactrackings (heuristicas de decisión), preprocesando la red (path consistency) para reducir el dominio de las variables, etc. (Sadeh96, Belhadji98, etc.) En el proceso de búsqueda de la solución, si una instanciación parcial no cumple alguna de las restricciones y no quedan valores en el dominio a instanciar, se fuerza un backtracking o vuelta atrás, eliminando así una parte del espacio de búsqueda. Una adecuada ordenación de variables provocará que los “fallos” se desplacen hacia arriba en dicho árbol, mientras que una buena ordenación de valores hace que la solución (si existe) pueda encontrarse más rápidamente. Sin embargo, su complejidad puede ser todavía elevada, debido al gran número de veces que usualmente se necesita volver hacia atrás. Finalmente, el algoritmo devuelve, si existe, una solución consistente. Si el problema no tiene solución debemos explorar todos los valores de todas las variables, con lo cual la complejidad del algoritmo se convierte, experimentalmente, en NP.

4.- UN PROCESO DE CLAUSURA - CSP.

Para mejorar la eficiencia del algoritmo de backtracking se incorpora el proceso de clausura, que determina (hace explícitas) el conjunto de restricciones que se pueden inferir de las explícitamente asertadas. Tipos de clausura

incluyen los algoritmos de arco-consistencia y *k* consistencia (Freuder82), (Mackworth77), variaciones del algoritmo "edge-finding" (LePape96), y shaving, (Carlier94).
Es posible, sin necesidad de utilizar CSP, encontrar la solución (si es que la hay), aplicando sucesivamente un proceso de clausura (propagación), para cada restricción del problema. Esto nos daría como resultado el conjunto de soluciones posibles, pudiendo obtener cualquier solución particular sin necesidad de hacer backtracking (Dechter92).
En el apartado 2, se ha indicado que utilizaremos una red de puntos temporales con restricciones métrico-disyuntivas. Si el proceso de clausura de una red de *n* nodos es *completo*, se deducen todas las posibles restricciones. Cuando es *correcto*, se obtienen solamente las restricciones factibles derivadas de las que ya se encuentran en la red. Así, una clausura completa y correcta obtendría una red mínima. Este proceso tiene un coste $O(n^3)$, cuando es un problema STP (Simple Temporal Problem), en el que únicamente tenemos restricciones no disyuntivas. Si la red es disyuntiva, como ocurre en el caso de scheduling, obtener el conjunto mínimo de soluciones tiene un coste de $O(e^{3n}k^{3n})$, siendo *e* el número de restricciones asertadas, *n* el número de nodos, y *k* el máximo número de disyunciones en una restricción (Dechter91). La alternativa propuesta nos permite obtener soluciones del problema, minimizando el número de backtrackings realizados. Ademas, podremos detectar en etapas iniciales del proceso si el problema es irresoluble.
Para ello, proponemos combinar de forma efectiva las técnicas CSP con técnicas de Razonamiento Temporal para resolver el problema de scheduling, reduciendo el número de backtrackings, y dirigiendo la búsqueda hacia una solución lo más cerca posible de la óptima.
Partimos de un conjunto de restricciones (*LC*), expresadas según hemos visto en el apartado 2. El método construye de forma iterativa la red de restricciones, aplicando cada vez un algoritmo de clausura. En primer lugar consideramos la duración de las actividades, precedencias, *ready time*, y *due time* para cada trabajo. Esto se puede conseguir con un coste temporal $O(n^3)$, puesto que en estas restricciones no se tienen disyunciones (Dechter91) y resulta un *STP*.
A continuación añadimos las restricciones disyuntivas. Un proceso de clausura completo, nos permitiría eliminar la necesidad de hacer backtracking. Sin embargo, cada vez que se añade una disyunción supone (en el peor de los casos) duplicar el número de soluciones alternativas, resultando un coste computacional muy elevado si las mantenemos todas.
Por ello, podría optarse por mantener solo una de las dos disyunciones, en base a un criterio heurístico. Así, el coste temporal se mantendría $O(n^3)$, pero podríamos necesitar hacer backtracking para encontrar la solución, si la elección realizada no fuese la adecuada. Además, es decisivo el orden de decisión sobre la disyunción a mantener, tal y como ocurre en *CSP*.
El método propuesto complementa esta alternativa. Las dos disyuntivas actualizadas en cada nueva restricción se mantienen, en el proceso de clausura, si no se dispone de suficiente información heurística para decidirse sólo por una de ellas. De esta forma se reduce la posiblilidad de hacer backtracking, pero se incrementa la complejidad del proceso. Para tomar decisiones efectivas nos basaremos en la información proporcionada por heurísticas locales y globales. En definitiva, el hecho de posponer las decisiones de ordenación de las operaciones, permite que mantengamos las mejores soluciones y descartemos las que no van a ser óptimas.

4.1. Detalle del proceso de Clausura-CSP

El proceso construye la solución de forma incremental. Con cada nueva inserción de una restricción, la red se modifica añadiendo nuevos arcos, o reemplazando las restricciones existentes por otras más restrictivas. El algoritmo se muestra

```
Función clausuraCSP;
    Crear el conjunto de restricciones temporales:
    LC ={t_i c_ij t_j};
    Seleccionar y propagar las restricciones no
        disyuntivas c_ij ∈ LC;
    Si detectamos una inconsistencia
        entonces terminamos; Fin si;
    Mientras LC ≠ ∅ hacer
        Limitar el final del scheduling c_TO,TF ;(h1)
        Detectar y propagar posibles restricciones no
         disyuntivas c_ij ∈ LC
        Si detectamos una inconsistencia
            entonces terminamos; Fin si;
        Selecionar y propagar la siguiente restricción
            c_ij ∈ LC      ;(Heurística h2)
        Si detectamos una inconsistencia
            entonces terminamos; Fin si;
        Si tenemos suficiente información ;(h3,h4)
            entonces
                Seleccionamos y comprobamos orden
                    entre operaciones
                Revisamos decisiones pendientes
                Si detectamos una inconsistencia
                    entonces backtracking; fin Si
        fin Si
    fin mientras
fin clausuraCSP;
```

Figura 1. Algoritmo propuesto para resolver el problema de scheduling

en la **Figura 1**. El proceso de clausura está basado en (Barber98). Este proceso obtiene la clausura total de un conjunto de restricciones métrico-disyuntivas, etiquetando cada disyuntiva y manteniendo un conjunto de etiquetas inconsistentes, para eliminar así disyunciones que no conducen a ninguna solución.

Inicialmente se introducen las restricciones no disyuntivas del problema, cuyo proceso de clausura podría detectar si el problema no tiene solución. En caso contrario, se efectúa un proceso iterativo.

El proceso de clausura infiere en cada momento la relación existente entre **T0** y **TF** (que representa el *makespan*). Cuánto mayor sea el tiempo disponible para terminar los trabajos, el número de soluciones posibles aumenta y, por lo tanto, el espacio de búsqueda. Por ello, en cada iteración tratamos de *restringir* heurísticamente (**h1**) el *makespan*. Lo hacemos calculando, para cada disyuntiva pendiente $((o_i, o_j) \acute{U} (o_j, o_i))$, el incremento máximo de makespan que ocasionaria, dada por $D_{xy}=max(dt(j_y)+ dur(o_i), dt(j_x) + dur(o_j))$, donde $dt(j_y)$ es el *due-time* (ver apartado 2) derivado del proceso de clausura, para el trabajo j_y, del que forma parte la operación o_j. Y añadimos una nueva restricción $(\boldsymbol{T0} \; [0 \; \mathring{a}(D_{xy})] \; \boldsymbol{Tf})$, que eliminará soluciones no óptimas.

Adicionalmente, y teniendo en cuenta que disponemos de la red mínima, podemos detectar ordenaciones imposibles entre las operaciones que comparten recursos, y eliminar las correspondientes disyunciones de las restricciones pendientes.

Para seleccionar la siguiente restricción a añadir del conjunto LC, que se asimila a un proceso de ordenación de variables, utilizamos la heurística **h2**. Esta heurística está basada en la holgura de la ordenación o *slack* (Smith & Cheng 93).

Cada operación tiene un tiempo más temprano de comienzo $est(o_i)$, restricción mínima entre *T0* y $on(o_i)$, y un tiempo más tardío de finalización, restricción máxima entre *T0* y $off(o_i)$. Se define la *holgura resultante de la secuenciación de* o_i *después de* o_j como el tiempo libre restante después de la ejecución de o_j a continuación de o_j : $slack(o_i, o_j)=lft(o_j)–est(o_i)–(dur(o_i)+dur(o_j))$. Cuanto mayor sea dicha holgura, hay más probabilidad de que puedan secuenciarse con éxito posteriores ordenaciones que afecten a o_i y a o_j . Por lo tanto elegiremos, del conjunto de restricciones pendientes de ordenar, aquella cuyo *slack* sea mínimo: $\min_{(u,v)}\{\min\{\min\{slack(o_u, o_v), slack(o_v, o_u)\}\}\}$, por considerar dicha ordenación más crítica, permitiéndonos detectar conflictos lo antes posible. En este proceso hay que tener en cuenta que siempre disponemos de la red mínima.

A continuación debemos decidir cual es la ordenación (disyuntiva) concreta (o_i, o_j) o bien (o_j,o_i), a elegir. Aquí es donde se decide mantener ambas soluciones activas o bien, eliminar una de ellas. La idea es aplicar distintas heurísticas para obtener la mejor información posible y tomar una decisión acertada. Concretamente hemos utilizado dos heurísticas. La primera de ellas (**h3)** tiene carácter global y escoge la ordenación que minimiza la relación propagada entre los puntos **T0** y **Tf**[4]. Este criterio de decisión intenta optimizar la solución. La segunda heurística (**h4**) tiene carácter local, y elige aquella ordenación cuyo *slack*, sea máximo, de forma que quede más "espacio libre" para secuenciar posteriores operaciones. Este criterio intenta evitar backtrackings. Si las decisiones tomadas por **h3** y **h4**, se contradicen, o no deciden, podemos dejar la elección pendiente, que será revisada en posteriores aserciones (iteraciones) del proceso. De esta forma estamos dirigiendo la búsqueda hacia la mejor solución de forma flexible, variando la "anchura" de la misma en función de la información disponible hasta el momento.

5.- EVALUACIÓN Y RESULTADOS PRELIMINARES.

El coste del algoritmo varía en función de sus parámetros. Se trata de encontrar un equilibrio entre el aumento de coste que supone la propagación de restricciones, que finalmente no pertenecerán al conjunto mínimo de soluciones, y el que supondría hacer backtrackings si se eliminasen incorrectamente soluciones necesarias u óptimas. Si siempre se efectúa una decisión, tendríamos una complejidad $O(n^3)$ para los procesos de clausura. Suponiendo que nunca decidimos, el coste sería $O(e^{3n} k^{3n})$, como se ha visto en el apartado 4. En los experimentos realizados, hemos comprobado que el algoritmo efectúa una decisión correcta en la mayoría de los casos, lo que conlleva una reducción considerable del número de backtrackings realizados, manteniendo un nivel de indecisiones computacionalmente aceptable.

Hemos probado algunas instancias ampliamente utilizadas como banco de pruebas de algoritmos de scheduling, a cuya creación han contribuido varios autores[5]. Concretamente analizaremos las conocidas como *ft06*, *la01*, *la02*, *car1* y *car2*. Las tres primeras son instancias *job shop*, y las dos

[4] Precisamente introducimos el punto temporal especial **Tf**, para poder aplicar las heurísticas globales propuestas (**h1** y **h3**), que aprovechan la información global derivada de la clausura. En cada iteración podemos extraer una cota superior e inferior del *makespan*, observando la restricción temporal existente entre **T0** y **Tf**.

[5] ftp://ftp.mscmga.ms.ic.ac.uk/pub/jobshop1.txt y ftp://ftp.mscmga.ms.ic.ac.uk/pub/flowshop1.txt

últimas son *flow shop* scheduling. Los resultados se muestran en la **Tabla 1**.
Para cada instancia se indica el número de restricciones disyuntivas que se generan, el número de veces que las heurísticas **h3** y **h4** han tomado la decisión acertada, el número máximo de indecisiones permitidas (si este número es 0 se decide siempre), el número de backtrackings realizados, así como el número de backtrackings evitados (si solamente hubiésemos utilizado heurísticas *CSP*), y finalmente la solución obtenida, y la óptima para cada instancia.

Instancia **(*m´n*)**	Nº Restricc. disyuntivas	*h3*	*h4*	Nº Máx. indecis. mantenidas	Sólución obtenida	Sól. óptima	Nº back. realizados	Nº back. evitados
ft06 (6×6)	90	90	0	0	55	55	0	0
la01 (10×5)	225	207	200	0	741	666	0	0
		207	200	1	701		0	17
		221	203	2	666		0	25
la02 (10×5)	225	210	203	0	734	655	0	0
		215	207	1	677		0	20
		221	215	2	655		0	22
c*ar1* (11×5)	275	258	214	0	7894	7038	0	0
		269	230	1	7326		0	32
		271	239	2	7038		5	34
c*ar2* (13×4)	312	284	250	0	7632	7166	0	0
		300	299	2	7166		0	50

Tabla 1. Resultados de los experimentos realizados con varias instancias.

La heurística ***h3*** propuesta ha sido capaz de tomar una decisión correcta, en la mayoría de los casos, contribuyendo a la obtención de una solución óptima. La heurística ***h4*** ha contribuido a decidirnos por una solución, aunque nos alejásemos más del óptimo buscado. La heurística ***h1*** se ha aplicado en todos los casos, comprobándose que en pocos pasos, acota el límite superior del scheduling, tanto más cuantas más restricciones hayamos añadido. La heurística ***h2*** también se ha aplicado en todos los casos y, como se puede comprobar en los backtrackings evitados, si se utilizase solamente junto con ***h4***, se producirían muchos más backtrackings.

6.- CONCLUSIONES Y TRABAJOS FUTUROS.

Se ha presentado una alternativa para resolver problemas de scheduling, mediante la *integración* de procesos de clausura de restricciones temporales y técnicas CSP. Esta integración permite detectar la insatisfabilidad del problema en etapas previas y obtener el conjunto mínimo de soluciones, minimizando la necesidad de backtracking. La aproximación propuesta intenta conducir la búsqueda hacia valores óptimos usando heurísticas de una forma flexible, manteniendo soluciones mientras no tengamos información suficiente para hacer elecciones correctas, pudiendo parametrizar el número máximo de indecisiones permitidas en cada iteración. Esto permite mantener la complejidad computacional del problema a unos niveles razonables, a costa de incrementar la posibilidad de efectuar backtracking.
Los resultados preliminares obtenidos muestran que el algoritmo mantiene indecisiones en pocos casos, haciendo así un buen uso de las heurísticas propuestas. Además, las indecisiones pendientes se resuelven en pocas iteraciones, de forma que el coste temporal y espacial no crece de forma continuada, sino que se auto limita según se va añadiendo más información.
Como trabajo futuro, se propone ampliar la tipología de las restricciones a tratar (asignación de recursos, duraciones de uso variables, introducción del coste de uso de los mismos, fabricación por lotes, etc.). También se pretende desarrollar nuevas heurísticas que hagan más eficiente el proceso de clausura, o incluso mejorar ésta. Una extensión interesante del algoritmo es dotarlo de capacidad de "aprendizaje", de forma que modifique de forma dinámica la cantidad de indecisiones permitidas. Finalmente, se está completando la implementación de un traductor de especificaciones a alto nivel de problemas de scheduling, tal que las restricciones temporales que constituyen la entrada del proceso se obtengan directamente a partir de dicha especificación.

REFERENCIAS.

Beck, J.C., and Fox, M.S. (1998): A generic framework for constraint-directed search and scheduling. *AI Magazine,* 101-130

Barber, F. (1998) Reasoning on complex temporal constraint. *Research Report DSIC-II/17/1998.* U.P.de Valencia.

Belhadji, S. and Isli, A. (1998): Temporal constraint satisfaction techniques in fob shop scheduling problem solving. *Constraints* 3 (2-3) 203-212

Carlier, J. and Pinson, E. (1994): Adjustment of heads and tails for the job-shop problem. *European Journal of Operations Research* 78 146-161

Dechter, R. (1991): Temporal constraint networks *Artificial Intelligence* 49 61-95

Dechter, R. (1992): From local to global consistency. *Artificial Intelligence* 55 87-107

Fox, B.R., and McMahon, M.B. (1991) Genetic operators for sequencing problem, *in G.J.E. Rawlings (ed.). Foundations of genetic algorithms* 284-300

Fox M.S. and Smith S.F. (1984): Isis: a knowledge-based system for factory scheduling. *Expert systems* 1 (1)

Freuder, E. (1982):A sufficient condition for backtrack-free search. *Journal of the ACM* 29 (1) 24-32

Freuder, E. (1989): Partial constraint satisfaction. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence.* Menlo Park, Calif.: American Association for Artificial Intelligence, 278-283

Garey, M.R. and Johnson, D.S. (1979): Computers and intractability: a guide to the theroy of NP-completeness. *New York: Freeman*

Glover,F.,Laguna,M.,Taillard,E. and de Werra, D.(1993) Tabu Search. *Annals of Operations Research* 41 (1) 4-32

Johnson, S. M., (1954): Optimal two-and three-stage production schedules with set-up times included. *Nav. res. Logist. Quart.* 1 61-68

Jonsson,P.,Bäckström,C.(1998) A unifying approach to temporal constraint reasoning. *A I.*, 102. 143-155

Jain, A.S. and Meeran S. (1999): Deterministic job-shop scheduling: past, present and future. *To appear in European Journal of Operational Research* 113 (2)

LePape, C. and Baptiste, P. (1996): Constraint-propagation techniques for disjunctive scheduling: the preemptive case. *Twelfth European Conference on Artificial Intelligence, 11-16 August,* Budapest, Hungary.

LePape, C. and Smith, S.F. (1988): Management of temporal constraints for factory scheduling. *Temporal aspects in Information Systems.* Elsevier Science Publischer B.V. (North-Holland).

Mackworth, A.K. (1977): Consistency in networks of relations. *Artificial Intelligence* 8 99-118

Mackworth, A.K. and Freuder, E. (1985): The complexity of some polynomial network-consistency algorithms for constraint-satisfaction problems. *Artificial Intelligence,* 25 65-74

Sadeh, N. and Mark, S. Fox, (1996): Variable and value ordering heuristics for the job-shop scheduling. constraint satisfaction problem. Artificial Intelligence 86(1): 1-41

Sadeh, N, (1991): Look-ahead techniques for micro-opportunistic job shop scheduling. *PhD. Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.*

Smith, S.F., and Cheng, C.C. (1993): Slack-based heuristics for constraint-satisfaction scheduling. *In Proceedings of the 11th National Conference on Artificial Intelligence. Menlo Park, Calif.:AAAI,* 139-144

Zweben, M., Davs, E., Daun, B. and Deale, M.J. (1993): Scheduling and rescheduling with iterative repair. *IEEE Transactions on Systems Man, and Cymernetics,* 23 (6) 1588-1596