ALGORITMO SATSA DE DOS PASOS PARA JSSP.

AUTORES

Marco Antonio Cruz Chávez
00334858@academ01.mor.itesm.mx

Arnulfo Martínez Perete
00371320@academ01.mor.itesm.mx

Noviembre del 2000

MATERIA

ANÁLISIS DE ALGORITMOS

PROFESORES.

Dr. Juan Frausto Solís
Dr. Fernando Ramos Quintana.

RESUMEN.

En este trabajo se presenta la documentación del algoritmo SATSA de dos pasos. En el primer paso se realiza una codificación de JSSP a SAT, se aplica una reducción de la codificación para transformar el problema tipo 3-SAT a 1-SAT. En el segundo paso la solución obtenida por el primer paso, se mejora con la meta-heurística de Recocido Simulado.

ALGORITMO SATSA DE DOS PASOS PARA JSSP.

1. CODIFICACION SAT.

Para la codificación a SAT del problema de Job shop Scheduling (JSP), primero se plantea este en forma de un modelo CSP para aprovechar la estructura especial que da CSP, en particular el planteamiento de las restricciones de capacidad de recursos que se involucran en Job shop [Smith 1993].

Un problema de satisfacción de restricciones (CSP) consiste en determinar si un conjunto de restricciones definidas a través de ecuaciones puede ser satisfecho. Cada restricción debe tener una forma que sea fácil de evaluar, de manera que cualquier dificultad de solucionar el problema viene de las interacciones que existen entre las restricciones y la necesidad de encontrar una asignación para las variables que las integran, la cual simultáneamente las satisfaga [Gu, et. al., 1997].

Stephen Smith representa el problema de Job shop scheduling como CSP:

- **Restricciones de precedencia:** Para cada relación de precedencia $i \rightarrow j$ especificada entre dos operaciones i, j dentro del proceso de un trabajo dado J

$$s_i + p_i \leq s_j \quad \textbf{(1)}$$

Donde p_i es el tiempo de procesamiento requerido por la operación i del trabajo J.

- **Restricciones de capacidad de recursos:** Cuando dos operaciones i y j requieren utilizar el mismo recurso

$$s_i + p_i \leq s_j \quad \vee \quad s_j + p_j \leq s_i \quad \textbf{(2)}$$

- **Tiempos de descarga r_J y tiempos límites d_J (Ready times y deadlines):** para cada operación i del trabajo J. Aquí r_J y d_J son asociados al trabajo J.

$$r_J \leq s_i \quad \textbf{(3)}$$

$$s_i + p_i \leq d_J \quad \textbf{(4)}$$

Una solución a un problema simple de Jobshop scheduling en CSP representado por este grupo de restricciones, es una asignación consistente de valores para las variables de tiempo de inicio s_i para cada operación i.

Para la codificación del modelo anterior a SAT. Por la forma en que está el modelo CSP y para hacer más fácil la codificación, se asignan como variables independientes a los tiempos de inicio de cada operación, de aquí que se deban de obtener los tiempos de inicio de cada operación para una asignación ya que para dos operaciones consecutivas i, j en un recurso compartido, se tiene que decidir si i se asigna antes que j o bien j se asigna antes que i. Para cada par de operaciones (i, j), en cada restricción, se realiza un mapeo al

conjunto booleano ={verdadero, falso} por lo que se introducen las siguientes variables booleanas [Crawford 1994]:

CSP	Traducción SAT	Significado	Cláusula
$s_i + p_i \leq s_j$	$pr_{i,j} = verdadero$	i precede a j	**(5)**
$s_i + p_i \leq s_j \quad \vee \quad s_j + p_j \leq s_i$	$pr_{i,j} \quad _V \quad pr_{j,i} = verdadero$	i precede a j o j precede a i	**(6)**
$r_J \leq s_i$	$sa_{i,ri} = verdadero$	i comienza al tiempo de liberación de J (es decir r_i) o más adelante	**(7)**
$s_i + p_i \leq d_J$	$eb_{i,di} = verdadero$	i termina al tiempo de término d_J o antes	**(8)**

Las cláusulas (5) a (8) se denomina conjunto C o conjunto de restricciones. Generalizando estas cláusulas y la formulación del problema, [Crawford 1994] determinó el siguiente conjunto S de coherencias mostrado en (.9) a (11) y que completa la codificación en SAT. El modelo es del tipo $C \wedge S$ que describe un conjunto de soluciones para JSP, que mediante valores propuestos en el inicio de las variables p_{ij} se puede resolver el problema SAT encontrado y determinar un schedule factible. Para evaluar las coherencias, estas requieren estar en forma normal conjuntiva (FNC). En S t significa el tiempo en que la operación puede comenzar o terminar.

Coherencias	FNC	Cláusula
$sa_{i,t} \rightarrow sa_{i,t-1}$	$\neg sa_{i,t} \vee sa_{i,t-1}$	**(9)**
$eb_{i,t} \rightarrow eb_{i,t+1}$	$\neg eb_{i,t} \vee eb_{i,t+1}$	**(10)**
$sa_{i,t} \rightarrow \neg cb_{i,t+pi}$	$\neg sa_{i,t} \vee \neg cb_{i,t+pi}$	**(11)**
$sa_{i,t} \vee pr_{i,j} \rightarrow sa_{j,t+pi}$	$\neg sa_{i,t} \vee \neg pr_{i,j} \vee sa_{j,t+pi}$	**(12)**

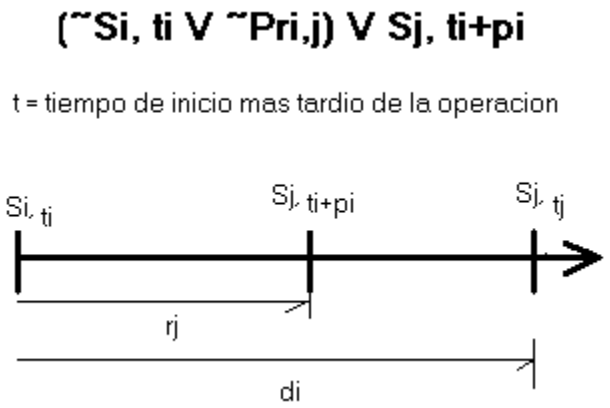
Como es evidente esta formulación es una versión del problema de decisión (NP-Completo), por lo cual solo se podrá comprobar si una asignación sugerida de tiempos de inicio de operación s_i es satisfactible (solución que no se conocerá si es la mejor). En el caso de buscar la versión de optimización (NP-Hard), se requerirá tratar de obtener la mejor asignación de tiempos de inicio en los que se minimice el tiempo de termino mas tarde encontrado de una operación (Ci = si + pi) y que no es otra cosa que el Makespan. Esto dentro del vasto campo de soluciones que se tengan en cada JSP.

Cuando en una codificación SAT de un problema algunas de sus cláusulas son siempre verdaderas y por lo tanto no intervienen en la solución, estas pueden ser eliminadas y al conjunto reducido de cláusulas se le da el nombre de codificación SAT reducida. Para

este método de solución las restricciones de capacidad de recursos, establecen el conjunto de variables independientes, mientras que las restricciones de precedencia, son datos del problema que se tienen que cumplir. Los ready times y los deadlines se asumen que siempre se cumplen siempre y cuando las cláusulas de capacidad de recursos sean verdaderas mediante su evaluación a través de los tiempos de inicio mas tardíos de las operaciones (explicado mas adelante). De esta forma, haciendo una asignación al conjunto de restricciones de capacidad de recursos se obtiene un grafo conjuntivo a partir del cual se pueden establecer los tiempos de inicio más tardíos para cada operación (el tiempo más tarde en el cual la operación puede comenzar para esa asignación).

Encontrando entonces el tiempo de inicio más tardío para cada operación, es posible determinar los tiempos de liberación y de término. Usando los valores de los tiempos de liberación y de término encontrados, éstos serán siempre verdaderos, de modo que en cada cláusula disyuntiva donde aparezcan, esta cláusula será trivialmente verdadera. Como consecuencia, el conjunto de cláusulas presentado de (2.5) a (2.12) será reducido solo a las cláusulas (2.12).

En la siguiente figura se puede entender porque los tiempos de liberación y de termino máximo para cada operación son siempre verdaderos si las cláusulas de capacidad de recursos son verdaderas:



Como se observa en la figura, para un par de operaciones *i, j* que compiten por un recurso y contando con una secuencia para su análisis de satisfactibilidad, donde se tiene que *i* precede a *j*. Por el calculo de los tiempos de inicio mas tardíos t_i y t_j de *i, j* respectivamente, el tiempo en el que finaliza *i*, es $ti+pi$ (ver figura), si t_j resulta ser mayor o igual que el termino de la operación *i*, entonces la cláusula $(\neg sa_{i,t} \vee \neg pr_{i,j} \vee sa_{j,t+pi})$ es verdadera. En caso de cumplirse lo anterior los ready times siempre serán verdaderos para cada operación, de la figura, r_j (r_j = tiempo de termino de *i*) jamás será violado puesto que la operación *j* iniciara al tiempo t_j (la operación *j* debe de inicial a partir de que termine *i*). De igual manera, los deadlines siempre serán verdaderos, como se muestra en la figura, para la operación *i*, esta puede retrasar su terminación (d_i) hasta el inicio de *j*, de aquí que entre el fin de *i* y el inicio de *j* puede existir un tiempo de ocio en el que la maquina no realice operación alguna (intervalo entre $S_{j,\ ti+pi}$ y $S_{j,tj}$), si este ocio no existe entonces d_i será el inicio de *j*.

El tiempo de inicio más tardío de una operación se determina a través de la ruta critica (ruta más larga desde el inicio a la operación en cuestión). El tiempo de inicio más

tardío de una operación se conforma con la suma de los tiempos de procesamiento de las operaciones sobre la ruta crítica sin tomar su propio tiempo. Conociendo los tiempos de inicio más tardíos de cada operación, se revisa la satisfactibilidad solo del conjunto de cláusulas (12) para cada par de operaciones.

La interpretación que se le da a (12) en forma de implicación es: Asegurar que si la operación *i* inicia a o después del tiempo *t* y además *j* sigue de *i*, entonces *j* no podrá iniciar hasta que *i* haya finalizado. Puesto que tenemos los tiempos de inicio podemos evaluar las variables $Sa_{i,t}$ y $Sa_{j,t+pi}$, donde t representa el tiempo de inicio encontrado para la operación i. Sustituyendo en (12) el tiempo de inicio para *i*, resulta que la primera de estas dos variables es siempre verdadera, mientras que la segunda dependerá de que $t + pi$ no rebase el tiempo de inicio previamente encontrado para *j*. Siendo pr_{ij} supuesta verdadera, el valor de verdad de la implicación (12) dependerá solo de su lado derecho. Entonces, para un par de operaciones (*i,j*), cuyos tiempos de inicio y de procesamiento conocemos, el lado derecho de (15) es falso, solo si el tiempo de inicio de *j* (i.e, $t + pi$) es mayor al tiempo de inicio calculado mediante el proceso de ruta critica.

Se propone una secuencia de operaciones de forma aleatoria y se obtienen los tiempos de inicio de cada operación, a continuación se prueba la satisfactibilidad de la fórmula FNC reducida, en caso de ser falsa se vuelve a proponer otra secuencia de operaciones hasta encontrar una formula reducida satisfactible. Esta solución obtenida (tiempos de inicio de las operaciones y sus secuencias) no es la óptima, simplemente es una asignación valida encontrada. Si se desea ver el problema como optimización combinatoria, entonces será requisito el proponer varias asignaciones que al evaluarse por SAT sean validas y de aquí se elija la mejor a través de una comparación en la obtención del Makespan de cada asignación propuesta. De aquí, podemos ver que estamos frente a un algoritmo no determinístico, debido a que proponemos de forma aleatoria varios conjuntos de asignaciones, las cuales tendrán que probarse una por una como factibles o in factibles.

2. RECOCIDO SIMULADO.

En cada iteración *k* de SA existe un schedule del problema, el cual se trata de que sea mejor que la iteración *k-1*. El proceso de búsqueda para encontrar un schedule que sea el optimo, es buscar en una vecindad compuesta por vecinos que pertenecen al schedule propuesto inicialmente (SC_o), la vecindad estará compuesta de los schedules obtenidos por permutación de trabajos que componen al mismo. Estas vecindades pueden ser pequeñas o grandes según el proceso de permutación que sea utilizado. El proceso de búsqueda para encontrar el optimo, hace que se mueva la búsqueda de un *SC* a otro. Del schedule en la iteración *k* (SC_k), la búsqueda se conduce dentro de su vecindad para obtener uno nuevo *SC*, este *SC* candidato se acepta de acuerdo al algoritmo de Metrópolis. Si *SC* es un mejor schedule que SC_k , entonces $SC_{k+1} = SC$. Si *SC* es mejor que el mejor obtenido hasta ahora SC_o, entonces se almacena $SC_o = SC$. Si SC es peor que SC_k entonces se acepta con una probabilidad dada por la función de boltzmann. De lo anterior se puede notar que los movimientos a peores soluciones se permiten para poder escapar de óptimos locales y

poder encontrar mejores soluciones. De la función de boltzmann se permite que si un *SC* vecino es significativamente muy malo, la probabilidad de aceptación es muy baja y el movimiento no es probable que se realice. En la práctica son varios los criterios de paro que se realizan. Uno es el correr el procedimiento hasta un numero determinado de iteraciones. Otro, es correr el procedimiento hasta que no existan mejoras en la solución. La figura 1 muestra el algoritmo para JSSP.

```
npert=1;
datos=inicializar(c,T,Tf,coef temp,BD,mc,MBD)
While(T > Tf){
 While(npert < 10){
    perturbacion(nc,BDT,datos);
    If(nc > c){
            E = nc - c;
            p = e^-DE/T;
            al = num(0.0,…,1.0);
            if(al < p){
                    c = nc;
                    BD = BDT;
   }
    else{
             c = nc;
             BD = BDT;
              if(c < mc){
                    mc = c;
                    MBD = BDT;
              }
    }npert++;
 }T=Coef_temp*T;
}
```

BD = Tiempos de inicio

BDT = Tiempos de inicio temporal

c = Costo

mc = Mejor costo

nc = Nuevo costo

T = Temperatura.

Tf = Temperatura final

Coef temp = Coeficiente temperatura.

npert = num de perturbaciones para alcanzar equilibrio termodinamico en cada temperatura.

Al inicio de RS

T = c

Figura 1 Algoritmo de Recocido Simulado para JSSP

De la figura 1 Para minimizar el Makespan. Se inicializan los datos de entrada. *MD* es el schedule que se toma como inicio, obtenido por medio de algún procedimiento anterior, y que se mejorara con el procedimiento de SA, almacenando el mejor continuamente en *MBD*. Al comienzo la temperatura inicial toma el valor del Makespan del schedule inicial y se decrementa en cada paso de acuerdo al coeficiente de temperatura. Cuando se efectúa la perturbación del schedule para encontrar a un vecino el cual se almacena en *BDT*, este vecino puede ser aceptado de acuerdo al algoritmo de Metrópolis.

2.1. Propuesta de perturbación.

Existen muchos esquemas de perturbación [Laarhoven et al, 1992]. Un esquema de perturbación propuesto para obtener vecinos de un schedule y que da buenos resultados utilizado con SA para aproximaciones al óptimo es el siguiente:

1.Elección aleatoria de un Trabajo = *J*.
2.De *J* se elige O_i con menor tiempo de inicio (S_i).
3.$O_k = P_i + S_k$. $k = 1,…nop$.
4.O_i, $S_i = 0$ (permutación)
5.Se obtiene un schedule activo (permutación).

Con la ayuda del ejemplo mostrado en la figura 2 de un JSSP de 2 x 3. En el primer paso del algoritmo, se elige de forma aleatoria un trabajo del schedule de inicio ($J = 3$). En el paso dos, se encuentra la operación O_i de J con el menor tiempo de inicio. En el paso tres, se suma a cada operación del sistema (con excepción de O_i) el tiempo de procesamiento P_i de la operación O_i, esto hace que todas las operaciones se muevan a la derecha un tiempo P_i. En el paso cuatro la operación O_i se permuta dándole un tiempo de inicio de cero (se pasa al inicio del schedule). Finalmente, en el paso cinco, una vez perturbado el schedule original, se obtiene un schedule activo el cual podrá ser mejor o peor que el original.

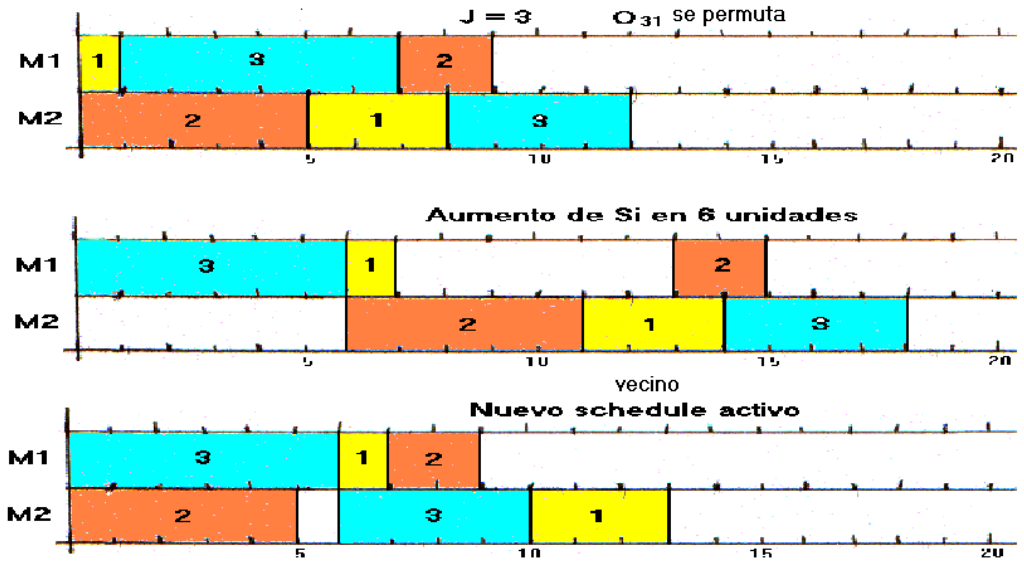


Figura 2 Esquema de perturbación para la obtención de vecinos.

REFERENCIAS.

[Crawford 1994] J.M. Crawford and A.B. Baker, “Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems”, Computational Intelligence Research Laboratory, 1994.

[Gu, et. al., 1997] J.Gu, P.W. Purdom, John Franco and B. W. Wah, "Algortthms for the satisfiability (SAT) problem: a survey",

[Laarhoven et al, 1992] Van Laarhooven, P. J. M., Aarts, E. H. L. and Lenstra, J. K. (1992) Job Shop Scheduling by Simulated Annealing, Operations Research, Jan-Feb, 40(1), 113-125.

[Smith, 1993] S. F. Smith, and C. C. Cheng, 1993. Slack-Based heuristics for constraint satisfaction scheduling. In Proccedings of the Eleventh National Conference on Artificial Intelligence, 134-144.