SECUENCIACIÓN DE TAREAS DE ENSAMBLADO CON RECURSOS LIMITADOS MEDIANTE ALGORITMOS DE EXPLORACIÓN DE ENTORNOS*

Joaquín Bautista, Raúl Suárez, Manuel Mateo, Amaia Lusa

Instituto de Organización y Control de Sistemas Industriales
Universidad Politéctica de Cataluña – Av. Diagonal 647, planta 11, 08028 Barcelona
{bautista, suarez, mateo, lusa}@ioc.upc.es

RESUMEN

La secuenciación de tareas con limitación de recursos (RCPSP*: resource-constrained project scheduling problem*) es un problema combinatorio clásico que aparece en las líneas de manufactura y producción. El problema consiste en establecer el orden de fabricación de las tareas y la asignación de recursos que suponga menor tiempo de ocupación del sistema. Las heurísticas greedy basadas en reglas de prioridad son las más empleadas para su resolución. Se han propuesto procedimientos de búsqueda local con definiciones clásicas de vecindario válidas para cualquier problema de secuencias. Aquí se proponen procedimientos de búsqueda local con definiciones de vecindario sobre el espacio de heurísticas y el de datos que tienen en cuenta las peculiaridades del problema; entre ellas un procedimiento de escalado y un algoritmo genético.

Palabras clave: RCPSP, Programación de actividades, heurísticas, búsqueda local, algoritmos genéticos, secuenciación de tareas.

---

* Este trabajo ha sido parcialmente subvencionado por los proyecto TAP98-0494 y TAP98-0471

1. INTRODUCCIÓN

La secuenciación de tareas con limitación de recursos (RCPSP: *Resource-Constrained Project Scheduling Problem*) es un problema clásico de optimización combinatoria que ha sido ampliamente tratado en la literatura (1), (2), (3). El problema consiste en determinar el programa compatible de menor duración de un conjunto de tareas con duraciones conocidas que presentan dos tipos de restricciones: potenciales y acumulativas.

En este contexto, se entiende por programa: establecer un calendario (instantes de inicio y finalización de las tareas) y una asignación de recursos. Si además el programa satisface las restricciones o ligaduras potenciales (precedencias o sucesiones obligadas entre tareas) y las ligaduras acumulativas (que la utilización de recursos a lo largo del tiempo no sea superior a sus disponibilidades) diremos que el programa es compatible. El objetivo es, por tanto, establecer el orden de ejecución de las tareas para obtener un programa compatible de menor duración: minimizar el instante de finalización de la última tarea (*makespan*). Obviamente, entre las múltiples aplicaciones del RCPSP se encuentra la secuenciación de tareas de emsamblado.

Para la resolución exacta del problema se han utilizado distintos procedimientos basados en *branch and bound* y en la programación dinámica (4), (5), aunque su aplicación sólo es viable para ejemplares de dimensiones pequeñas debido al carácter NP-hard del problema (6).

Para resolver ejemplares de tamaño real se han empleado diversas heurísticas, entre las que destacan las *greedy* basadas en el empleo de reglas de prioridad que condicionan el lanzamiento de las tareas, ya sea en serie o en paralelo (7), (8). Las reglas se refieren o combinan aspectos tales como la duración, la holgura total, el número de actividades siguientes, la solicitud de recursos, etc. y sirven para establecer una ordenación de tareas en cada iteración, con el propósito de seleccionar la mejor según la regla, entre un conjunto de candidatas compatibles con la parte de la solución ya construida; las tareas candidatas son aquéllas cuyas precedentes han sido programadas y su requerimiento de recursos no supera las disponibilidades de los mismos. Usualmente, cada heurística de este tipo tiene asociada una sola regla, y la regla determina (salvo que se emplee el sorteo) la tarea que debe lanzarse en cada paso.
La aplicación de este tipo de algoritmos heurísticos suele ofrecer soluciones aceptables, en promedio, tanto más cuanto mayor sea el número de aspectos que combina la regla. No obstante, no puede concluirse que exista una única regla que supere a todas las demás ante cualquier ejemplar de problema. Por otra parte, salvo que se incorpore el azar al procedimiento, siempre ofrecerán las mismas soluciones.

Una segunda clase de heurísticas son los métodos de búsqueda local o procedimientos de exploración de entornos (9), entre ellos se encuentran: los procesos de escalado (HC), recocido simulado (SA), búsqueda tabú (TS) y los algoritmos genéticos (GA). Esta segunda clase de heurísticas proporciona vías alternativas para buscar soluciones en un espacio delimitado por la definición de un vecindario. No obstante, si la definición del vecindario es general y válida para cualquier problema de secuencias, se pierde entonces el conocimiento sobre el problema específico que sí es tenido en cuenta por las heurísticas *greedy*.

En el presente trabajo se proponen procedimientos de búsqueda local que incorporan los aspectos positivos de ambos tipos de procedimiento: 1) el conocimiento sobre el RCPSP que ofrecen las reglas de prioridad específicas del problema y 2) la posibilidad deseable en todo problema combinatorio de generar soluciones en el espacio de búsqueda.

2. HEURÍSTICAS DE BÚSQUEDA LOCAL

Los métodos de búsqueda local se emplean para explorar entornos constituidos por soluciones vecinas. Una forma típica de definir el vecindario o conjunto de soluciones vecinas a una concreta consiste en caracterizar una solución mediante una secuencia de elementos (p.e. una secuencia de tareas de ensamblado que establece el orden en que deben realizarse) y realizar intercambios de elementos entre posiciones de la secuencia siguiendo unas reglas. Está claro que este tipo de definición de vecindario es general y no explota la información específica del problema a resolver, aunque ofrece la ventaja de ser universal y aplicable a todo problema se secuencias.

Se han propuesto otras alternativas para la definición de vecindarios (10) basadas en la relación que existe entre una heurística h y la solución s que se obtiene al aplicarla a un ejemplar del problema p: $h(p) = s$. Esta relación permite establecer vecindarios en el espacio de los problemas y en el espacio de las heurísticas.

La definición de vecindarios en el espacio de los problemas se basa en la idea de perturbar aleatoriamente los datos del problema, dentro de unos límites razonables, y después aplicar una heurística para obtener una solución; lógicamente, la solución se evalúa posteriormente con los datos originales.

Por su parte, la clave para definir vecindarios en el espacio de las heurísticas está en la posibilidad de desarrollar nuevas versiones parametrizadas del conjunto de heurísticas disponibles y específicas de un problema. Esta última alternativa se puede desarrollar al menos de dos formas distintas en el RCPSP:

1. Definir una nueva regla híbrida mediante una combinación lineal ponderada del conjunto de reglas originales disponibles de lanzamiento ρ_i: $\rho = \sum_{\forall i} \pi_i \rho_i$
2. Dividir las decisiones de secuenciación en grupos o ventanas y asociar a cada ventana una regla. Esta segunda vía sugiere la posibilidad de caracterizar una solución a través de un vector de reglas: $r = (\rho_{[1]}, \rho_{[2]}, ..., \rho_{[N]})$

donde N es el número de tareas y $\rho_{[k]}$ es la regla que se aplica en la k-ésima decisión de secuenciación.

3. UN EJEMPLO ILUSTRATIVO

En la figura 1 se muestra un conjunto formado por 12 componentes que se deben montar por una pareja de robots de idénticas características; las ligaduras de precedencia son evidentes (p.e. B1 precede a C1) y las duraciones para las tareas B1, C1, D1, B2, C2, D2, E1, F1, G1, E2, F2 y G2 son respectivamente, 6, 7, 6, 15, 17, 14, 8, 7, 9, 10, 10 y 11 s.

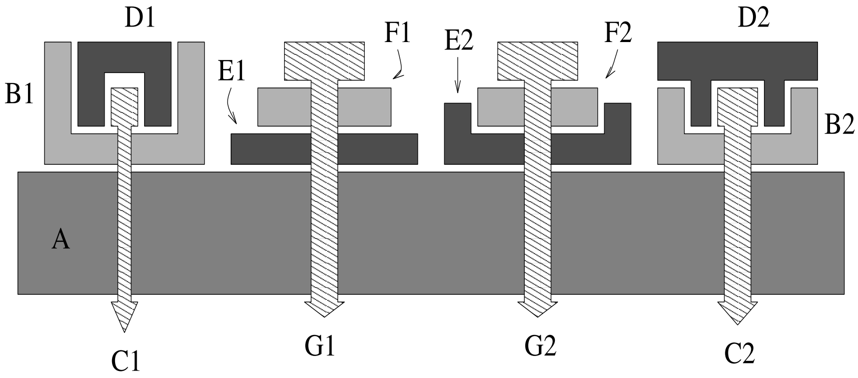

Figura 1: Ensamblado mediante dos robots.

El ejemplo propuesto se ha resuelto mediante un procedimiento de lanzamiento de tareas en paralelo con las 100 reglas de prioridad que se relacionan en el anexo 1: SIO (*Shortest Imninent Operation*), GRD (*Greatest Resource Demand*), *Minimum Job Slack* (MINSLK), *Wheighted Resource Utilitation Ratio and Precedence* (WRUP), etc. La mejor solución hallada (sin optimización local) presenta un tiempo total de ensamblado de 62 s. y se obtiene mediante las reglas 4-13, 27, 51-59, 61-70, 83, 89, 91, 92, 96 y 100; la aplicación del resto de reglas conduce a soluciones comprendidas entre 63 s. y 71 s. (regla 97). No obstante, es fácil hallar a simple vista soluciones óptimas para el ejemplar, tal como se muestra en la figura 2.

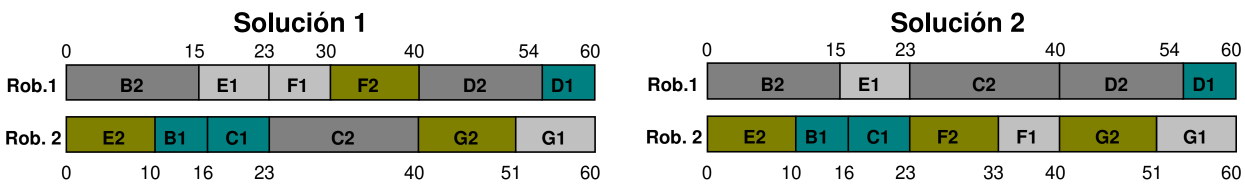

Figura 2: Soluciones óptimas para el ejemplo

Este sencillo ejemplo no pretende cuestionar la calidad de las reglas, sino la forma de aplicarlas y la rigidez de las heurísticas *greedy* a pesar de lo refinada que pueda ser la regla de prioridad empleada.

4. ALGORITMO BÁSICO DE SECUENCIACIÓN

Dado un vector de reglas *r*, la forma de obtener una solución es inmediata; para ello, basta el siguiente algoritmo:

Notación

N número de tareas
M número de tipos de recurso
i índice de tarea, $1 \leq i \leq N$
j índice de recurso, $1 \leq j \leq M$
k índice de decisión de programación, $1 \leq k \leq N$
T instante de lanzamiento
$P(i)$ duración de la tarea i
$C(i)$ instante de finalización de la tarea i programada
$\Gamma(i)$ conjunto de precedentes de i
$R(i,j)$ unidades de recurso j requeridas por la tarea i
$R(j)$ unidades de recurso j disponibles. Inicialmente $R_0(j)$
X conjunto de tareas pendientes de programar. Inicialmente X_0.
Y conjunto de tareas candidatas por precedentes programadas ($Y \subseteq X$)
Z conjunto de tareas programables por relaciones de precedencia y disponibilidad de recursos ($Z \subseteq Y$).
W conjunto de tareas en proceso
S conjunto de tareas en proceso con menor fecha de finalización
$\rho_{[k]}$ regla de la k-ésima decisión de programación.
C máximo de los instantes de finalización de las tareas programadas.
C_{max} *Makespan*

Algoritmo A_1

0. Inicializar:
   $T \leftarrow 0$; $k \leftarrow 1$; $C(i) \leftarrow \infty$ para todo i $(1 \leq i \leq N)$; $R(j) \leftarrow R_0(j)$ para todol j $(1 \leq j \leq M)$; $X \leftarrow X_0$; $W \leftarrow \varnothing$
1. Construcción de Y: $Y = \{y \in X : (\ C(x) \leq T \wedge x \in \Gamma(y)\ ) \vee \Gamma(y) = \varnothing\ \}$.
2. Construcción de Z: $Z = \{z \in Y : R(z,j) \leq R(j)$ para todo $j\}$; Si $Z = \varnothing$ Ir a 5.
3. Programar tarea:
   Ordenar Z según la regla $\rho_{[k]}$
   Sea z^* la primera tarea de Z
   $C(z^*) = T + P(z^*)$; $R(j) \leftarrow R(j) - R(z^*,j)$ para todo j; $W \leftarrow W + \{\ z^*\}$; $C = \max[C(w)]$ con $w \in W$
   $Y \leftarrow Y - \{\ z^*\}$; $X \leftarrow X - \{\ z^*\}$; Si $X = \varnothing$ Ir a 6.
4. Incrementar contador de decisiones: $k \leftarrow k+1$; Ir a 2
5. Liberar recursos:
   Buscar $S = \{\ s : C(s) = \min[C(w)]$ con $w \in W\ \}$
   $R(j \leftarrow R(j) + \Sigma_{s \in S} R(s,j)$ para todo j; $T \leftarrow T + C(s)$ con $s \in S$; $W \leftarrow W - S$; Ir a 1.
6. Determinar C_{max}: $C_{max} = \max[C(w)]$ con $w \in W$
   Finalizar.

La aplicación de $\boldsymbol{A_1}$ al ejemplo propuesto ofrece la solución 1 presentada en la figura 2, a partir, por ejemplo, de la secuencia de reglas r = (40,40, 7, 7, 7,40,40,40,40,40,40,40). En general, dado un vector de reglas r, y un algoritmos A, se puede definir una heurística h a partir del par (r, A): $h = h(r,A)$.

El algoritmo $\boldsymbol{A_1}$ puede ser la base para cualquier procedimiento de búsqueda local que permita generar soluciones vecinas en el espacio de las heurísticas; es decir, los intercambios, alteraciones, etc. se realizarán sobre las reglas, en lugar de realizarlas sobre las tareas.

5. PROCEDIMIENTOS DE EXPLORACIÓN EN EL ESPACIO DE HEURÍSTICAS

Caben varias alternativas para definir procedimientos de exploración en el espacio de heurísticas, aquí presentamos dos de ellas.

5.1. Proceso de escalado (*Hill Climbing*)

Es sin duda el más sencillo y se puede formalizar así:

Algoritmo HC_1

0. Inicializar: Crear una heurística incumbente sorteando para cada posición de la cadena de reglas un elemento del conjunto de reglas heurísticas.

Iterar L veces a través de los pasos siguientes:

1. Determinar al azar una posición de la cadena de reglas.
2. Asignar una regla, seleccionada al azar, a la posición determinada (regla distinta a la que posee la heurística incumbente).
3. Aplicar la nueva heurística para obtener la solución y calcular su *makespan*.
4. Si el *makespan* del paso 3 es inferior o igual al de la solución incumbente, hacer como heuristica incumbente la resultante del paso 2. Ir a 1

5.2. Algoritmo genético

A continuación se describe otro procedimiento para generar soluciones en el espacio de heurísticas bajo la forma de un algoritmo genético.

Notación

I número de elementos de la población
L número de generaciones
p instancia del problema a resolver
Π_r población de ascendentes de secuencias de reglas
Π_h población de ascendentes de heurísticas
Π_s población de ascendentes de soluciones
Δ_r población de descendientes de secuencias de reglas
Δ_h población de descendientes de heurísticas
Δ_s población de descendientes de soluciones
Λ_r población de descendientes mutados de secuencias de reglas
Λ_h población de descendientes mutados de heurísticas
Λ_s población de descendientes mutados de soluciones
Ω_r población de elementos regenerables de secuencias de reglas

r_i $\quad$ i-ésimo elemento de Π_r o Δ_r o Λ_r,o Ω_r
h_i $\quad$ i-ésimo elemento de Π_h o Δ_h o Λ_h o Ω_h
s_i $\quad$ i-ésimo elemento de Π_s o Δ_s o Λ_s o Ω_s

Algoritmo GA_1

0. Inicializar:
   0.1 Generar una población inicial Π_r de I secuencias homogéneas y distintas de reglas: $\Pi_r = \{ r_i = (\rho_{[1]},..., \rho_{[N]}) : \rho_{[1]} = .. = \rho_{[N]} \}$
   0.2 Definir la población inicial de heurísticas: $\Pi_h = \{ h_i = h_i(r_i,A_1) : r_i \in \Pi_r \}$
   0.3 Generar la población de soluciones de p y evaluar: $\Pi_s = \{ s_i = h_i(p) : h_i \in \Pi_h \}$
   0.4 Guardar como incumbentes el par (h^*,s^*) que presenta menor makespan.
   0.5 Determinar el fitnees de los elementos de Π_s. El fitness se calcula de la forma:

$$f_j = \frac{(D_j - \alpha D_{\min})^{-1}}{\sum_{i=1}^{I} (D_i - \alpha D_{\min})^{-1}}$$

   donde:

   D_i $\quad$ makespan de la i-esima solución
   $D_{\max}$ $\quad$ mayor makespan de la población
   $D_{\min}$ $\quad$ menor makespan de la población
   α $\quad$ índice de homogeneidad de la población: $\alpha = \frac{1}{I}\sum_{i=1}^{I}\frac{D_i - D_{\min}}{D_{\max} - D_{\min}}$

Iterar L veces a través de los pasos siguientes:

1. Selección de ascendentes: Constituir $I/2$ parejas de elementos de Π_r en función del fitness de los elementos de Π_s .
2. Selección de parejas a cruzar:
   2.1 Determinar la probabilidad de cruce de la generación en curso: $P_c = P_c(\alpha)$
   2.2 Asignar un número aleatorio a cada pareja de secuencias de reglas
   2.3 Decidir para cada pareja de secuencias de reglas si hay cruce o no en función de su número aleatorio y de P_c.
3. Generar descendientes:
   3.1 Cruzar las parejas de secuencias de reglas seleccionadas, generando dos descendientes por cada pareja como mínimo. Se crea Δ_r.
   3.2 Generar Δ_h y Δ_s a partir de Δ_r de forma análoga a 0.2 y 0.3, respectivamente.
   3.3 Determinar el makespan de los elementos de Δ_s .Si algún elemento de Δ_s presenta mejor makespan que el de la solución incumbente, guardar como heurística y solución incumbentes el par (h^*,s^*) asociado a dicho elemento.
4. Mutar descendientes:
   4.1 Determinar la probabilidad de mutación de la generación en curso: $P_m = P_m(\alpha)$
   4.2 Asignar un número aleatorio a cada elemento de Δ_r.
   4.3 Decidir qué elementos de Δ_r se mutan o no en función de su número aleatorio y de P_m.
   4.4 Mutar los elementos de Δ_r seleccionados. Se crea Λ_r.
   4.5 Generar Λ_h y Λ_s a partir de Λ_r de forma análoga a 0.2 y 0.3, respectivamente.
   4.6 Determinar el makespan de los elementos de Λ_s Si algún elemento de Λ_s presenta mejor makespan que el de la solución incumbente, guardar como heurística y solución incumbentes el par (h^*,s^*) asociado a dicho elemento.

5. Regeneración de la población:
   5.1 Construir la población de elementos regenerables: $\Omega_r \leftarrow \Pi_r + \Delta_r + \Lambda_r$
   5.2 Determinar el fitness de los elementos de las poblaciones Δ_s y Λ_s como se indica en 0.5.
   5.3 Seleccionar I elementos del conjunto Ω_r en función del fitness de los elementos de los conjuntos Π_s, Δ_s y Λ_s.

6. EXPERIENCIA COMPUTACIONAL

Se ha realizado un experimento con $\boldsymbol{GA_1}$ mediante la resolución de 270 ejemplares del problema con N (número de tareas) comprendida entre 6 y 15, con tres valores de M (número de tipos de recursos): 1, 2 y 3, y con valores de $P(i)$ (duración de las tareas) comprendidos entre 1 y 16. También se han considerado distintas densidades de grafo de Davis & Patterson (número de arcos dividido por numero de actividades), en concreto se han establecido tres agrupaciones: baja, media y alta con valores comprendidos entre [0.50, 1.25], [1.25, 1.75] y [1.75, 2.50], para ejemplares de más de 10 actividades, y [0.00, 0.50], [0.50, 1.00] y [1.00, 1.50], para los de menos de 11 actividades. Para obtener los óptimos, $\boldsymbol{GA_1}$ empleó menos de 12 minutos con un Pentium II.

Para tener en consideración todas las reglas expuestas en el anexo 1, se ha tomado un tamaño de población I =100 (cada elemento representa una heurística asociada a una regla); para aumentar el tamaño de la población inicial, basta incorporar otras reglas, o generar reglas híbridas mediante combinaciones lineales ponderadas de reglas simples.

Por otra parte, los elementos de Π_r se seleccionan por sorteo, para constituir parejas, en función del fitness; las parejas se forman a partir de dos extracciones consecutivas. En cuanto al cruce y mutación de elementos, se han tomado las probabilidades, respectivas, $P_c = 1 - 0.5\alpha$ y $P_m = 0.05 + 0.95\alpha$, dependientes del índice de homogeneidad. El cruce entre dos elementos se realiza por dos puntos seleccionados al azar; mientras que el proceso de mutación incorporado presenta tres variantes: blanda (selección de dos genes al azar de un individuo e intercambiar sus informaciones), media (aplicar a un gen, tras su selección al zar, todas las reglas y retener el cromosoma que ofrece menor makespan) y dura (2-intercambio exhaustivo entre las informaciones de los genes). Finalmente, el proceso de regeneración funciona de forma similar al de selección.

7. CONCLUSIONES

El presente trabajo propone una vía para explorar soluciones en el problema RCPS. La clave del método consiste en incorporar al procedimiento de busqueda local el conocimiento que aportan las heurísticas específicas del problema. De esta forma se puede caracterizar una solución mediante una secuencia de reglas de prioridad. Para ilustrar el método se ha descrito un HC y un AG cuya explotación ha ofrecido resultados satisfactorios.

ANEXO 1

Nomenclatura:

$P(i)$ duration of task i.
$R(i,j)$ number of units of resource j required by task i.
$R_0(j)$ number of available units of resource j
Z set of tasks satisfying the precedence constraints and resources availability.
$ns(i)$ number of direct successors
$nst(i)$ number of successors
$i \to h$ h is a direct successor of i.
$i \Rightarrow h$ h is a successor of i.
EST Earliest Start Time.
LST Latest Start Time.
EFT Earliest Finish Time.
LFT Latest Finish Time.

Secuenciar la tarea z^* : $v(z^*)=\max_{i\in Z}[v(i)]$

Nombre	**Regla**
1.SIO *Shortest Imminent Operation*.	$v_1(i) = -P(i)$
2. GRD *Greatest Resource Demand*	$v_2(i) = P(i)\sum_{j=1}^{M} R(i,j)$
3. GRPW *Greatest Rank Positional Weight*.	$v_3(i) = P(i)\sum_{i\Rightarrow h} P(h)$
4-14. WRUP *Weighted Resource Utilization Ratio and Precedence*	$v_4(i) = w_p ns(i) + w_r \sum_{j=1}^{M} \frac{R(i,j)}{R_0(j)}$ $w_p = 1 - w_r;\quad w_r = (0.0, 0.1, ..., 0.9, 1.0)$
15-25. WRUP2	$v_5(i) = w_p \sum_{i\to h} P(h) + w_r \sum_{j=1}^{M} \frac{R(i,j)}{R_0(j)}$
26. ALEA.	$v_6(i) = Random(i)$
27. MTS *Most Total Successors*	$v_7(i) = nst(i)$
28-38. WRUP3	$v_8(i) = w_p P(i) + w_r \sum_{j=1}^{M} \frac{R(i,j)}{R_0(j)}$
39-49. WRUP4	$v_9(i) = w_p P(i) + v_5(i) = w_p \sum_{i\to h} P(h) + v_8(i)$
50-60. WRUP5	$v_{10}(i) = w_p nst(i) + w_r \sum_{j=1}^{M} \frac{R(i,j)}{R_0(j)}$
61-71. WRUP6	$v_{11}(i) = w_p \sum_{i\Rightarrow h} P(h) + w_r \sum_{j=1}^{M} \frac{R(i,j)}{R_0(j)}$
72-82. WRUP7	$v_{12}(i) = w_p P(i) + v_{11}(i) = w_p \sum_{i\Rightarrow h} P(h) + v_8(i)$
83. MIT *Most Immediate Successors*	$v_{13}(i) = ns(i)$
84. MIT2	$v_{14}(i) = ns(i) + \sum_{i\to h} P(h)$
85. MIT3	$v_{15}(i) = ns(i)P(i)$
86. MIT4	$v_{16}(i) = ns(i)\left(P(i) + \sum_{i\to h} P(h)\right)$
87. MIT5	$v_{17}(i) = ns(i) + P(i)\sum_{j=1}^{M} R(i,j)$
88. MIT6	$v_{18}(i) = ns(i)P(i)\sum_{j=1}^{M} R(i,j)$
89. MIT7	$v_{19}(i) = nst(i) + \sum_{i\Rightarrow h} P(h)$
90. MIT8	$v_{20}(i) = nst(i)P(i)$
91. MIT9	$v_{21}(i) = nst(i)\left(P(i) + \sum_{i\Rightarrow h} P(h)\right)$

92. MIT10	$v_{22}(i) = nst(i) + P(i)\sum_{j=1}^{M} R(i,j)$
93. MIT11	$v_{23}(i) = nst(i)P(i)\sum_{j=1}^{M} R(i,j)$
94. LST *Latest Start Time*	$v_{24}(i) = -LST(i)$
95. EST *Earliest Start Time*	$v_{25}(i) = -EST(i)$
96. LFT *Latest Finish Time*	$v_{26}(i) = -LFT(i)$
97. EFT *Earliest Finish Time*	$v_{27}(i) = -EFT(i)$
98. MINSLK *Minum Job Slack*	$v_{28}(i) = -(LST(i) - EST(i))$
99. RSM *Resource Scheduling Method*	$v_{29}(i) = -max\left[0, \min_{h \in Z-\{i\}} (EFT(i) - LST(h))\right]$
100. RSM2	$v_{30}(i) = -\sum_{h \in Z-\{i\}} max[0, (EFT(i) - LST(h))]$

REFERENCIAS

(1) Hartmann, S., Kolisck, R. (1998). Experimental Evaluation of State of Art Heuristics for thr Resource Constrained Project Scheduling Problem. *Wp. IBUK*, No. 476.

(2) Özdamar, L., Ulusoy, G. (1995). Survey on the resource-constrained project scheduling problem. *IIE Transactions*, 27 (5), 574-586.

(3) Weglarz, J. Ed. (1998). *Handbook on Recent Advances in Project Scheduling*. Kluwer, Amsterdam.

(4) Patterson, J.H. (1984). A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Sc.*, 30 (7), 854-867.

(5) Simpson, W.P., Patterson, J.H. (1996). "A multiple-tree search procedure for the resource-constrained project scheduling problem". *EJOR*, 89, 525-542.

(6) Blazewicz, J., Lenstra, J.K., Rinnoy Kan, A.H.G. (1983). "Scheduling projects to resource constraints: Classification and complexity". *Discrete Applied Mathematics*, 5, 11-24.

(7) Álvarez-Valdés, R., Tamarit, J.M. (1989). Heuristic algorithms for a resource constrained project scheduling: A review and an empirical analysis", *Advances in Project Scheduling*, 113-134. R. Slowinski, J. Weglarz (Ed.). Elsevier, Amsterdam.

(8) Kolisch, R. (1996). Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, 14, 179-192.

(9) Díaz, A., Glover, F. & Ghaziri, H. & González, J.M. & Laguna, M. & Moscato, P. & Tseng, F. (1996). *Optimización heurística y redes neuronales*. Paraninfo.

(10) Storer, R.H., Wu, S.D., Vaccari, R. (1992). New search spaces for sequencing problems with application to Job Shop Scheduling". *Management Sc.*, 38 (10), 1495-1509.