Un algorisme genètic en l'espai d'heurístiques per a cèl·lules de muntatge amb recursos limitats

Joaquín Bautista **Raúl Suárez** **Manuel Mateo** **Amaia Lusa**

Institut d'Organització i Control de Sistemes Industrials (UPC).

Diagonal 647, planta 11, 08028 Barcelona. {bautista,suarez,mateo,lusa}@ioc.upc.es

Resum

La programació de tasques amb limitació de recursos, quan n'hi ha múltiples (RCPSP, de *Multiple Resource Constrained Project Scheduling Problem*) és un problema combinatori clàssic en línies de producció i muntatge. Consisteix en establir l'ordre de fabricació de tasques i l'assignació de recursos per ocupar el sistema al mínim. Per a resoldre'l s'utilitza força sovint procediments heurístics *greedy*, o sigui, "sense fre", basats en regles de prioritat. Així, s'han proposat diversos procediments de cerca local per buscar-ne solucions acceptables. Aquest treball proposa l'ús d'un algorisme genètic per generar la solució, amb procediments de cerca local per definir solucions en un veïnat d'heurístiques i de dades.

Paraules clau: *genetic algorithms*, *scheduling*.

1 Introducció

El problema de programació de projectes amb múltiples recursos (RCPSP) s'ha tractat àmpliament per nombrosos autors (com a exemples, [8][11][16]). Si es volen assolir solucions exactes, poden trobar-se mitjançant procediments *branch-and-bound*, de branca més fita, així com amb programació dinàmica [13][14]. No obstant, aquests procediments són només útils en cas de problemes amb unes dimensions reduïdes, ja que la seva complexitat és del tipus *NP-hard* [2].

Per tal de resoldre problemes reals, s'han anat proposant diferents heurístiques com, per exemple, aquelles que es basen en regles de prioritat considerant una limitació en la programació de les tasques en sèrie o en paral·lel [1][9]. Les regles consideren diferents aspectes com temps de fabricació (durada de les activitats), marges de programació, nombre de tasques subsegüents, necessitats de recursos, aleatorietat, etc. Les regles s'apliquen pas a pas per escollir una tasca entre un conjunt, quan les precedents de les quals ja han estat programades, i la necessitat de recursos encaixa en la disponibilitat dels mateixos.

Normalment, cada heurística d'aquest tipus s'ha associat a tan sols una regla tal que determini la tasca a programar en cada situació (menys si s'usa una selecció aleatòria).

Aquestes heurístiques sovint proporcionen solucions acceptables, i quants més aspectes es consideren en una regla, millor en serà la solució en mitjana. Malgrat tot, hom no pot concloure que hi hagi una regla millor que qualsevol altra per a qualsevol instància del problema. I encara més, si no fos per la incorporació d'una selecció aleatòria, les regles sempre acaben produint les mateixes solucions.

Un altre tipus d'heurístiques es basen en la cerca local [3] com, per exemple, Procés d'Escalada (HC de *Hill Climbing*), Recuita Simulada (SA *de Simulated Annealing*), Cerca Tabú (TS de *Tabu Search*) i Algorismes Genètics (GA de *Genetic Algorithms*). Els GA, introduïts per Holland el 1975 [7], es poden aplicar en el camp de l'optimització per a diversos problemes combinatoris [5] i, en particular, per a problemes de programació, i així resoldre el mateix problema [4][10] o bé analitzar el comportament de heurístiques [6][12].

Aquest segon grup d'heurístiques proporciona una sèrie d'alternatives a l'hora de buscar solucions en un veïnat definit. Malgrat tot, no solen considerar un coneixement específic d'un problema si el veïnat ha estat definit de manera general. Això no passa, en canvi, amb heurístiques *greedy*.

En el present treball, es proposa un procediment de cerca local incloent els aspectes positius d'ambdós tipus d'heurístiques: 1) el coneixement sobre el RCPSP que ofereixen les regles de prioritat del problema i, 2) la possibilitat de generar solucions en l'espai de cerca. A tal efecte, les solucions es caracteritzen per seqüències de regles de prioritat. Cada seqüència de regles genera una o més solucions seguint un algorisme força simple que optimitza el temps total necessari per a la realització totes les tasques, temps conegut com a *makespan*. En concret, s'aplica un algorisme genètic que genera les solucions usant creuaments, mutacions i la regeneració sobre diverses seqüències amb diferents regles de prioritat.

Aquest treball ha estat subvencionat pels projectes TAP98-0494 i TAP98-0471.

2 Heurístiques d'exploració d'entorns

Els mètodes de cerca local (com TS, SA, GA, etc.) s'usen en el camp de l'optimització per explorar un veïnat de solucions. Una manera habitual de definir els veïnats en un problema de programació consisteix en considerar l'intercanvi de tasques a realitzar en la programació d'activitats. Aquest és un procediment general que en cap moment fa servir informació específica sobre el problema.

Altres aproximacions a la definició d'un veïnat usen la relació entre una heurística h i la solució s obtinguda quan s'aplica h al problema p, és a dir, $h(p) = s$ [15]. Aquesta relació permet determinar veïnats en l'espai dels problemes i en l'espai de les heurístiques.

Per tal d'obtenir un veí en l'espai de problemes, es fan les següents accions: 1) s'introdueix una pertorbació aleatòria (dins d'un cert rang) en dades del problema (pot ser, com a exemple, canviar en un 10% la durada del temps de muntatge de cada part), 2) una heurística específica s'aplica sobre les noves dades per obtenir una solució "fictícia" (és a dir, una seqüència fictícia per muntar totes les parts), 3) s'avalua la solució "fictícia" (calculant el *makespan*) amb les dades originals del problema.

La definició de veïnats en l'espai de les heurístiques s'aconsegueix amb variacions parametritzades sobre el conjunt d'heurístiques específiques del problema. Això es pot fer, com a mínim, de dues maneres en l'RCPSP:

1) Definint una nova regla híbrida ρ com a combinació lineal de les regles de programació originals ρ_i, és a dir, $\rho=\Sigma_{\forall i}\ \pi_i\ \rho_i$.
2) Dividint la programació en subconjunts ordenats de regles (com, per exemple, que les tres primeres tasques siguin programades per la regla #2, les següents dues per la regla #7, etc.). Un cas extrem d'aquesta visió s'esdevindrà quan cada decisió de la programació estigui caracteritzada per una regla particular, és a dir, per a un problema amb N tasques la programació estaria controlada pel vector $r = (\rho_{[1]}, ...,\rho_{[k]}, ..., \rho_{[N]})$, on $\rho_{[k]}$ és la regla aplicada en la decisió k (per bé que la regla $\rho_{[N]}$ és irrellevant, s'inclou per homogeneïtat).

Una primera fase consistí en la recollida d'heurístiques diferents, aptes per determinar l'assignació de tasques als diferents recursos limitats. El procediment implementat es basava en la programació de tasques en paral·lel utilitzant 100 conegudes regles, que es troben en l'Annex A. El llistat de regles inclou regles com, per exemple, la regla que escull l'operació imminent més curta (SIO, *Shortest Imminent Operation*), la que considera la major demanda d'unitats de recurs (GRD, *Greatest Resource Demand*), la que considera alhora ponderats les precedències de tasques prèvies i el ratio d'ús d'un recurs (WRUP, *Weighted Resource Utilization Ratio and Precedence*), la que opta per l'operació programable de mínim marge (MINSLK, *Minimum Job Slack*), entre d'altres.

3 Algorisme de Programació de Tasques

3.1 Algorisme Bàsic (BS)

Donat un vector de regles $r = (\rho_{[1]}, ...,\rho_{[k]}, ..., \rho_{[N]})$, la solució de programació de tasques s'obté directament usant el següent algorisme **BS**.

Nomenclatura:

N	nombre de tasques (components a muntar).
M	nombre de tipus de recursos (robots, etc.).
i	índex de tasca, $1 \leq i \leq N$.
j	índex de recurs, $1 \leq j \leq M$.
k	índex de decisió en programació, $1 \leq k \leq N$.
T	temps de programació.
$P(i)$	durada de la tasca i.
$C(i)$	temps d'acabament de la tasca i.
$\Gamma(i)$	conjunt de precedències de tasques de la tasca i.
$R(i,j)$	nombre d'unitats de recurs j necessàries per a la tasca i.
$R(j)$	nombre d'unitats de recurs j disponibles (inicialment $R_0(j)$).
X	conjunt de tasques a programar (inicialment X_0).
Y	conjunt de tasques que satisfan les restriccions de precedència ($Y \subseteq X$)
Z	conjunt de tasques que satisfan les restriccions de precedència i la disponibilitat de recursos ($Z \subseteq Y$).
z^*	primera tasca de Z
W	conjunt de tasques que s'estan executant.
S	conjunt de tasques s'estan executant amb temps d'acabament més proper.
$\rho_{[k]}$	regla de decisió de programació k en el vector de regles r.
C	màxim temps d'acabament de taques programades.
$C_{\max}$	temps necessari per a completar totes les tasques (és a dir, *makespan*).

Inici BS

1. Inicialitzar:
   $T \leftarrow 0$
   $k \leftarrow 1$
   $C(i) \leftarrow \infty \quad \forall i\ (1 \le i \le N)$
   $R(j) \leftarrow R_0(j) \quad \forall j\ (1 \le j \le M)$
   $X \leftarrow X_0$
   $W \leftarrow \varnothing$
2. Crear Y:
   $Y = \{y \in X : (C(x) \le T\ \ \forall x \in \Gamma(y)) \vee \Gamma(y) = \varnothing \}$.
3. Crear Z:
   $Z = \{z \in Y : R(z,j) \le R(j)\ \forall j\}$
   SI $Z=\varnothing$ ANAR A 6
4. Programar la tasca:
   Obtenir Z segons la regla $\rho_{[k]}$
   $C(z^*) = T + P(z^*)$
   $R(j) \leftarrow R(j) - R(z^*, j)\ \forall j$
   $W \leftarrow W + \{ z^* \}$
   $C = \max[C(w)]$ amb $w \in W$
   $Y \leftarrow Y - \{ z^* \}$
   $X \leftarrow X - \{ z^* \}$
   SI $X=\varnothing$ ANAR A 7
5. Passar a la següent decisió:
   $k \leftarrow k+1$
   ANAR A 3
6. Relaxació de recursos:
   Buscar $S = \{ s : C(s) = \min[C(w)]$ amb $w \in W \}$
   $R(j) \leftarrow R(j) + \Sigma_{s \in S} R(s,j)\ \forall j$
   $T \leftarrow T + C(s)$ amb $s \in S$
   $W \leftarrow W - S$
   ANAR A 2
7. Determinar $C_{\max}$:
   $C_{\max} = \max[C(w)]$ amb $w \in W$

Fi BS

En general, donat un vector de regles r i un algorisme A es pot definir una heurística h a partir del parell (r, A), o sigui, $h=h(r, A)$.

L'algorisme **BS** es pot fer servir per a qualsevol procediment de cerca local que permeti generar solucions veïnes en l'espai de les heurístiques. Aleshores, s'alteraran les regles en lloc de les tasques.

3.2 Algorisme Genètic (GEN)

La generació de solucions en l'espai de les heurístiques (és a dir, amb vectors del tipus $r = (\rho_{[1]}, ..., \rho_{[k]}, ..., \rho_{[N]})$ que **BS** pugui utilitzar) s'ha realitzat usant l'algorisme heurístic **GEN**, que es descriu a continuació.

Nomenclatura:

I	nombre d'individus (vectors de regles) en la població.
L	nombre d'iteracions (generacions).
p	instància del problema a resoldre.
Π_r	població de progenitors de les seqüències de regles.
Π_h	població de progenitors de les heurístiques.
Π_s	població de progenitors de les solucions.
Δ_r	població de descendents de les seqüències de regles.
Δ_h	població de descendents de les heurístiques.
Δ_s	població de descendents de les solucions.
Λ_r	població de descendents mutats en les seqüències de regles.
Λ_h	població de descendents mutats en les heurístiques.
Λ_s	població de descendents mutats en les solucions.
Ω_r	població de seqüències de regles elegibles en la propera iteració (generació).
r_i	element i dels conjunts Π_r , Δ_r , Λ_r i Ω_r.
h_i	element i dels conjunts Π_h , Δ_h , Λ_h i Ω_h.
s_i	element i dels conjunts Π_s , Δ_s , Λ_s i Ω_s.

Inici GEN

1. Initialització:
   1.1 Generar la població Π_r inicial de I com a:
   $\Pi_r = \{ r_i = (\rho_{[1]}, ..., \rho_{[N]}) : \rho_{[1]} = ... = \rho_{[N]} \}$
   1.2 Generar la població inicial d'heurístiques:
   $\Pi_h = \{ h_i = h_i(r_i, \mathbf{BS}) : r_i \in \Pi_r \}$
   1.3 Generar la població de solucions de p i avaluar la seva durada global (*makespan*):
   $\Pi_s = \{ s_i = h_i(p) : h_i \in \Pi_h \}$
   1.4 Guardar el parell (h^*,s^*) amb millor durada global com a heurística i solució incumbents.
   1.5 Determinar l'aptitud (*fitness*) f_j dels elements de Π_s calculada com a:

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

amb:

D_i	durada global de la solució i
$D_{\max}$	durada global màxim de la població
$D_{\min}$	durada global mínim de la població
α	índex d'homogeneïtat de la població

$$\alpha = \frac{1}{I} \sum_{i=1}^{I} \frac{D_i - D_{\min}}{D_{\max} - D_{\min}}$$

2. Iterar L vegades seguint els passos següents:

   2.1. Selecció de progenitors:
   Construir $I/2$ parells d'elements de Π_r segons l'aptitud dels elements de Π_s.
   2.2. Elecció dels parells per al creuament:
   2.2.1. Determinar la probabilitat de l'actual creuament: $P_c = P_c(\alpha)$.
   2.2.2. Assignar un nombre aleatori a cada parell de seqüències de regles.
   2.2.3. Decidir, per a cada parell de seqüències de regles, si cal fer un creuament segons el nombre aleatori corresponent i P_c.
   2.3. Generació de descendents:
   2.3.1.Creuar el parell seleccionat de seqüències de regles per generar dos descendents, creant Δ_r.
   2.3.2. Generar Δ_h i Δ_s a partir de Δ_r com es va fer a 1.2 i 1.3 respectivament.
   2.3.3. Determinar la durada global dels elements de Δ_s Si qualsevol element de Δ_s té durada global millor que la solució incumbent, aleshores guardar el parell (h^*,s^*) associat a l'element com a heurística i solució incumbents.
   2.4. Mutació de descendents:
   2.4.1. Determinar la probabilitat de mutació de la generació actual: $P_m = P_m(\alpha)$.
   2.4.2. Assignar un nombre aleatori a cada element Δ_r.
   2.4.3. Decidir els elements que es mutaran de Δ_r segons el seu nombre aleatori i P_m.
   2.4.4. Mutar les elements escollits de Δ_r creant Λ_r.
   2.4.5. Generar Λ_h i Λ_s a partir de Λ_r com es va fer a 1.2 i 1.3 respectivament.
   2.4.6. Determinar la durada global dels elements de Λ_s. Si qualsevol element de Λ_s té un valor millor que la solució incumbent, aleshores guardar el parell (h^*,s^*) associat a aquell element com a heurística i solució incumbents.
   2.5. Regeneració de la població:
   2.5.1.Construir la població d'elements elegibles: $\Omega_r \leftarrow \Pi_r + \Delta_r + \Lambda_r$
   2.5.2. Determinar l'aptitud dels elements de les poblacions Δ_s i Λ_s com fou assenyalat a 1.5.
   2.5.3. Escollir I elements d'Ω_r segons l'aptitud dels elements de Π_s , Δ_s i Λ_s.

Fi GEN

En els següents subapartats s'aclareixen alguns aspectes particulars de l'algorisme implementat.

Població Inicial

La població considerada té una mida I =100 amb l'objectiu d'incloure el tractament de totes aquelles heurístiques derivades de les regles que hi ha en l'Annex A. Això vol dir que la població inicial es composa de 100 vectors de regles r_i = ($\rho_{[1]}$,..., $\rho_{[N]}$), cadascun amb una regla particular en tots els seus components $\rho_{[1]}$= ...= $\rho_{[N]}$.

Això permet l'exploració de totes aquelles solucions que es generen a partir de les heurístiques *greedy*, sense fre. Per tal d'incrementar la mida de la població inicial tan sols és necessari incloure noves regles o generar-ne d'híbrides, a partir de combinacions lineals de les anteriors regles.

També fou incorporada com a regla (regla 26) la selecció aleatòria de la tasca a programar en cada moment, per permetre la generació de qualsevol solució. L'algorisme inclou la selecció aleatòria si és necessari quan el conjunt de regles no garanteix la generació de tot l'espai de solucions.

Procés de Selecció

Els elements de Π_r es seleccionen aleatòriament amb major probabilitat per a aquells elements amb millors valors d'aptitud.

Probabilitats de Creuament i de Mutació

La probabilitat de qualsevol creuament o mutació, P_c i P_m respectivament, depèn de l'índex d'homogeneïtat α de la població en curs (segons el punt 1.5 de l'algorisme GEN).

Així doncs, una població amb força individus similars serà modificada per mitjà de mutacions ja que els creuaments no serien efectius per a la diversificació. Per als experiments s'han fet servir els següents valors:

$$P_c = 1-0.5\alpha, \text{ i } P_m = 0.05+0.95\alpha.$$

Procés de Creuament

Donats dos vectors de regles (els progenitors), es seleccionen dos components aleatòriament, i es fa un intercanvi de components (és a dir, un intercanvi de regles) entre els dos punts d'ambdós vectors, i s'obté dos nous vectors de regles (els descendents).

Procés de Mutació

S'han considerat tres tipus diferents de mutació d'un vector de regles: dèbil, mitjana i forta. La probabilitat de cada tipus de mutació ve donada a priori.

Mutació dèbil: es selecciona aleatòriament dos components del vector de regles i s'intercanvien.

Mutació mitjana: es selecciona aleatòriament una regla del vector, aquesta regla es va substituint successivament per a totes les regles disponibles i es selecciona la combinació que genera el mínim *makespan*.

Mutació dura: és equivalent a provar totes les possibles mutacions dèbils del vector de regles i seleccionar-ne aquella amb mínim *makespan*.

Procés de Regeneració

Es selecciona aleatòriament els elements de Ω_r donant prioritat superior a aquelles seqüències de regles amb millors aptituds.

4 Una aplicació pràctica: cèl·lules de muntatge

La figura 1 mostra un conjunt de 12 parts que s'han de muntar sobre la base A, amb dos robots del mateix tipus. Es tracta de fixar a la base 4 conjunts similars, formats cadascun per 3 elements, dos dels quals permeten la fixació del conjunt. La taula 1 recull el temps necessari (en segons) per al muntatge de cada part i les relacions de precedència entre les tasques a realitzar.

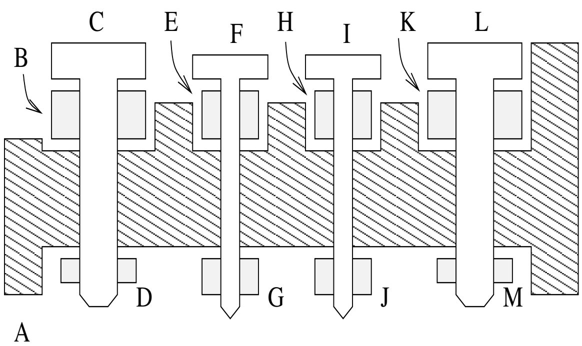

Figura 1: Muntatge que han de realitzar dos robots.

Tasca	Durada	Precedència
B	12	-
C	14	B
D	12	C
E	16	-
F	14	E
G	18	F
H	20	-
I	20	H
J	22	I
K	30	-
L	34	K
M	28	L

Taula 1

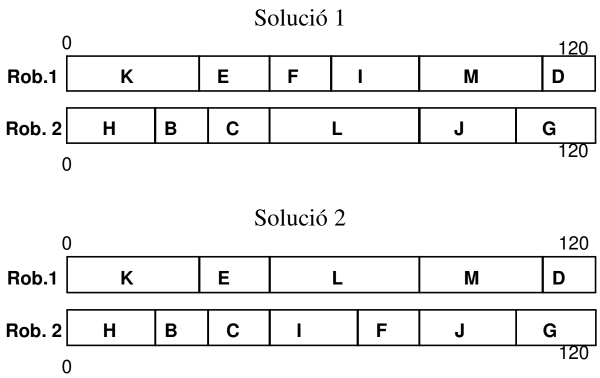

Figura 2: Dues solucions òptimes (de durada global=120 s) per al problema de la figura 1, amb les tasques de cada robot i el temps després de cada tasca.

La millor solució sense cap mena d'optimització local subsegüent té una durada global, o sigui *makespan*, de 124 segons, obtingut amb vectors únicament composats amb una qualsevol de les regles: 4-13, 27, 51-59, 61-70, 83, 89, 91, 92, 96 i 100. L'ús d'altres regles que no sigui aquestes produeix solucions entre 126 i 142 segons (el pitjor dels casos correspon a la regla #97, o sigui, $\rho_{[k]}$= regla#97 $\forall k$). Malgrat això, en aquest exemple es pot trobar fàcilment una solució òptima, com una de les que hi ha en la figura 2. Aquest simple exemple no implica que les regles considerades siguin poc apropiades, sinó que la manera com s'apliquen és el que no es pot considerar òptim.

Amb l'aplicació de l'algorisme **BS** es genera un vector de regles: r = (40,40,7,7,7,40,40,40,40,40,40,40), solució trobada durant la segona generació de la població que dóna la solució 1 de la figura 2. És interessant remarcar que si s'utilitza les regles #7 i #40 independentment, les solucions tarden 124 i 126 segons, respectivament.

5 Experimentació

Per tal de validar l'algorisme proposat, s'han escollit 270 instàncies diferents de problemes de programació de tasques amb recursos limitats, i s'han resolt amb un Pentium II 233MHz.

Els paràmetres considerats han estat:

- Nombre de tasques: $6 < N < 15$.
- Tipus de recursos (robots, etc.): $1 \leq M \leq 3$.
- Units d'un recurs *j* (nombre de robots): $2 \leq R_o(j) \leq 5$.
- Durada d'una tasca *i*: $1 \leq P(i) \leq 16$.
- 6 ratios diferents de #precedències/*N*.

L'algorisme proposat **GEN** va obtenir les solucions òptimes de les 270 instàncies del problema en menys de 12 minuts, amb l'equipament informàtic indicat.

Freqüència de les Regles

La freqüència amb què apareixen les 100 regles en les 270 solucions òptimes determinades queda reflectida en la figura 3. La freqüència, que inicialment per a cada regla era de l'1%, ha evolucionat, com indiquen els resultats, cap a l'aparició d'algunes regles amb una freqüència superior. En particular, és interessant d'observar el 6% que assoleix la regla 26 (que es correspon amb la selecció aleatòria d'una tasca).

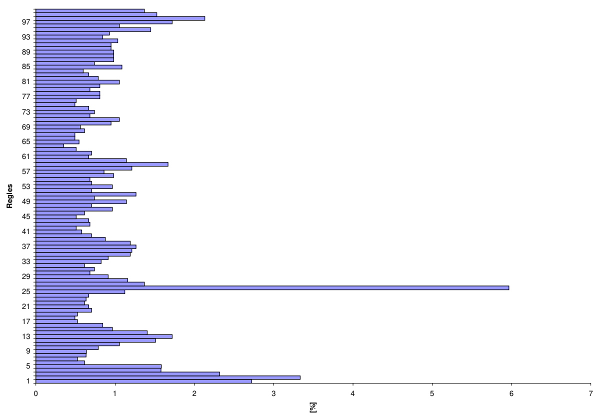

Figura 3: Freqüència final de les 100 regles en els experiments.

7 Conclusions

En aquest treball s'ha presentat un mètode de cerca de solucions per al problema RCPSP usant algorismes genètics aplicat, en aquest cas, a la programació d'operacions de muntatge amb recursos limitats (com són els robots). La principal aportació del mètode suposa la incorporació del coneixement específic que proporcionen heurístiques específiques per al problema en procediments de cerca local. En aquest sentit, la solució ve caracteritzada per una seqüència de regles de prioritat. Els resultats obtinguts en els experiments, força satisfactoris, constaten la seva validesa.

Annex A: Llistat de Regles

Nomenclatura:

$P(i)$	durada de la tasca *i*.
$R(i,j)$	nombre d'unitats del recurs *j* necessàries per a la tasca *i*.
$R_0(j)$	nombre disponible d'unitats del recurs *j*
Z	conjunt de tasques que satisfan les relacions de precedència i la disponibilitat de recursos.
$ns(i)$	nombre de successors directes.
$nst(i)$	nombre de successors.
$i \rightarrow h$	*h* és un successor directe d'*i*.
$i \Rightarrow h$	*h* és un successor d'*i*.
EST	*Earliest Start Time* (menor instant d'inici).
LST	*Latest Start Time* (major instant d'inici).
EFT	*Earliest Finish Time* (menor instant d'acabament).
LFT	*Latest Finish Time* (major instant d'acabament).

Programació de la tasca z^* : $v(z^*)=\max_{i\in Z}[v(i)]$

NOM	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.*	$v_4(i) = w_p ns(i) + w_r \sum_{j=1}^{M} \frac{R(i,j)}{R_0(j)}$
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	$v_{30}(i) = -\sum_{h \in Z-\{i\}} max[0, (EFT(i) - LST(h))]$

*Es considera una regla diferent per a cada valor de w_r que compleixi $w_r \in \{0, 0.1,...,0.9,1\}$, w_p =1- w_r .

Referències

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

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

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

[4] Fujimoto H., Yasuda K., Tanigawa Y. & Iwahashi K., "Application of Genetic Algorithm and Simulation to Dispatching Rule-Based FMS Scheduling". *1995 IEEE International Conference on Robotics and Automation*, Nagoya, Japan,V1 (1995), 190-195.

[5] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, Mass., 1995.

[6] Gyoung K. & Lee C.S.G., "An Evolutionary Approach to the Job-Shop Scheduling Problem". *1994 IEEE International Conference on Robotics and Automation*, San Diego, CA, (1994), 501-506.

[7] Holland, J.H., *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Mich., 1975.

[8] Kim, J. Desrochers A. & Sanderson, A., "Task Planning and Project Management using Petri Nets". 1995 *IEEE International Symposium on Assembly and Task Planning*, Pittsburgh, Pennsylvania, (1995), 265-271.

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

[10] Mori, M. & Tseng, C.C., "Genetic Algorithm for multi-mode resource constrained project scheduling problem". *European Journal of Operational Research*, 100 (1997), 134-141.

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

[12] Padman, R. & Roehrig, S.F., "A Genetic Programming Approach for Heuristic Selection in Constrained Project Scheduling", *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, (1997), 405-421. R.S. Barr & R.V. Helgason & J.L. Kennington (Ed.). Kluwer, Norwell, MA, USA.

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

[14] Simpson, W.P. & Patterson, J.H., "A multiple-tree search procedure for the resource-constrained project scheduling problem". *European Journal of Operational Research*, 89 (1996), 525-542.

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

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