Application of genetic algorithms to assembly sequence planning with limited resources

J. Bautista, A. Lusa, R. Suárez, M. Mateo, R. Pastor, A. Corominas

Institut d'Organització i Control de Sistemes Industrials (UPC)
Diagonal 647, planta 11, 08028 Barcelona , Spain. Phone: +34-934016653, Fax: +34-934016605.
Contact emails: bautista@ioc.upc.es, suarez@ioc.upc.es, mateo@ioc.upc.es

Abstract

Heuristic procedures based on priority rules are quite frequently used to solve the *multiple resource-constrained project-scheduling problem* (RCPSP), i.e. task programming with limited resources. The rules are based on the problem knowledge. Different local search procedures have been proposed in order to look for acceptable solutions in scheduling problems. In this work, local search procedures, that define the solution neighborhood based on greedy heuristics, are proposed to assign assembly operations to a fixed number of robots in a manufacturing cell. A genetic algorithm is used to generate the solution.

1 Introduction

The multiple resource-constrained project scheduling problem (RCPSP) has been extensively treated in the literature (e.g. [1][2][3]). Exact solutions have been obtained using branch-and-bound procedures as well as dynamic programming [4][5]. Nevertheless, these procedures are only useful for low dimension problems due to its NP-hard complexity [6].

In order to solve realistic problems, different heuristics have been used like, for instance, those based on priority rules constraining the serial or parallel dispatching of tasks [7][8]. The rules consider different aspects like, for instance, processing times (activity duration), slacks, number of subsequent tasks, resource requirements, randomizing, etc. The rules are applied step by step to choose a task among a set of them whose precedents have already been scheduled, while taking care that the resource requirements fit the available resources. Usually, each heuristic of this type has been associated to only one rule that determines the task to be dispatched at each situation (unless random selection is used).

These heuristics often produce acceptable solutions, and, as average, the higher the number of aspects considered in a rule the better the solution is. Nevertheless, it cannot be concluded that there exists one rule that works better than any other for any instance of the problem. Moreover, unless a random selection of tasks is incorporated, the rules will always produce the same solutions.

Another type of heuristics is based on local search [9] like, for instance, Hill Climbing (HC), Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithms (GA). The GA, introduced by Holland in 1975 [10], can be applied to the optimization of several combinatory problems [11] and, in particular, to the scheduling problems to analyze the behavior of different heuristics [12][13] as well as to solve the problem itself [14][15].

This second type of heuristics provides alternative ways to look for solutions in a defined neighborhood. Nevertheless, the particular knowledge of any scheduling problem is not considered if the neighborhood is defined in a general way. This does not happen with greedy heuristics.

In this work, a local search procedure is proposed including the positive aspects of both types of heuristics: 1) the knowledge about the RCPSP offered by the priority rules of the problem and, 2) the possibility of generate solutions in the search space. For this purpose, the solutions are characterized by sequences of priority rules. Each sequence of rules generates one or more solutions following a simple algorithm that optimize the makespan (total time needed for the real execution of the task). A Genetic Algorithm is applied to generate the solutions using crossovers, mutations and regeneration of different priority-rules sequences.

2 Local Search Heuristics

Local search methods (TS, SA, GA, etc) are used to explore a solution neighborhood. A typical way to define a neighborhood in a scheduling problem is the interchange of tasks. This is a general procedure that does not use the specific information about the problem.

Other approaches to the definition of a neighborhood use the relation between a heuristic h and the solution s obtained applying h to a problem p, i.e. $h(p) = s$ [16]. This relation allows the determination of neighborhoods in both the problem space and the heuristics space.

This work was partially supported by the CICYT Projects TAP 96-0868, TAP 98-0494 and TAP 98-0471.

In order to obtain a neighbor in the problem space the following actions are done: 1) introduce a random perturbation (within some range) in the data of the problem (e.g. change in a 10% the duration of the assembly time of each part), 2) one particular heuristic is applied to the new data to obtain a "dummy" solution (i.e. a dummy sequence to assembly all the parts), 3) the "dummy" solution is evaluated (i.e. the makespan is computed) with the original data of the problem.

The definition of neighborhoods in the heuristics space is done by developing parameterized variations of the set of specific heuristics of the problem. This can be done in at least two ways in the RCPSP:

1) Defining a new hybrid rule ρ as a linear combination of the original dispatching rules ρ_i, i.e. $\rho=\Sigma_{\forall i}\ \pi_i\ \rho_i$.
2) Dividing the dispatching into ordered subsets of rules (e.g. the three first tasks will be dispatched by rule #2, the next two tasks by rule #7, etc). An extreme case of this approach is that each decision in the dispatching is characterized by one particular rule, i.e. for a problem with N tasks the dispatching is controlled by the vector $r = (\rho_{[1]}, ...,\rho_{[k]}, ..., \rho_{[N]})$, where $\rho_{[k]}$ is the rule applied in the decision k (note that the rule.$\rho_{[N]}$ is irrelevant, but we include for homogeneity).

3 Assembly example

Figure 1 shows a set of 12 parts to be assembled on the base A by two robots of the same type (the parts Ci, Gi and Di $i\in\{1,2\}$ act as fasteners). Table 1 summarizes the time (in seconds) needed for the assembly of each part and the precedence relations between them.

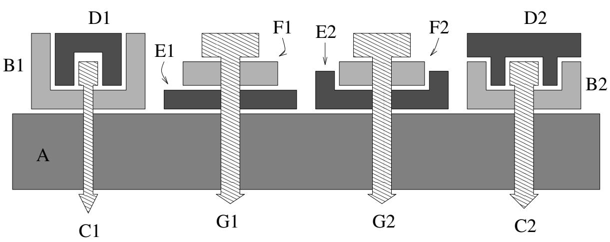

Figure 1: Assembly to be performed using two robots.

A first attempt to determine the assignment of tasks to each robot was done using a procedure based on the parallel dispatching of 100 well known rules. The set of rules, listed in Appendix A, includes, for instance, SIO (Shortest Imminent Operation), GRD (Greatest Resource Demand), Weighted Resource Utilization Ratio and Precedence (WRUP), Minimum Job Slack (MINSLK), among others.

The best solution without any subsequent local optimization has a makespan of 62 seconds, obtained with vectors uniquely composed by any of the rules: 4-13, 27, 51-59, 61-70, 83, 89, 91, 92, 96 and 100. The use of other rules produces solutions between 63 and 71 seconds (the worst case for rule #97, i.e. $\rho_{[k]}$ = rule_#97 $\forall k$). Nevertheless, in this example it is easy to find an optimal solution, like any of those in Figure 2. This simple example does not show that the rules are inadequate but that the way they are applied is not optimal.

Task	Duration	Precedence
B1	6	-
C1	7	B1
D1	6	C1
E1	8	-
F1	7	E1
G1	9	F1
E2	10	-
F2	10	E2
G2	11	F2
B2	15	-
C2	17	B2
D2	14	C2

Table 1

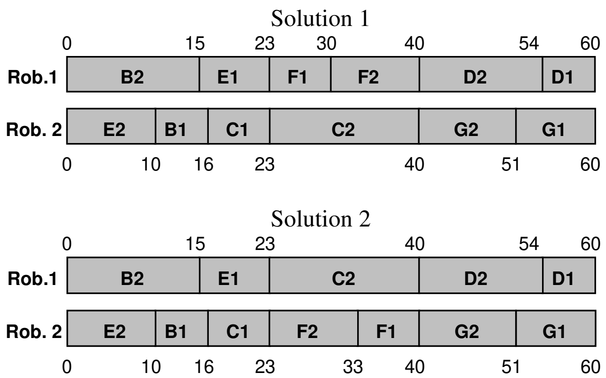

Figure 2: Two optimal solutions (makespan=60s) for the assembly problem of Figure 1 showing the task assigned to each robot and the total time after each assembly operation.

4 Basic Dispatching Algorithm

Given a vector of rules $r = (\rho_{[1]}, ...,\rho_{[k]}, ..., \rho_{[N]})$, the task scheduling solution is directly obtained using the algorithm **A1** described below.

Nomenclature:

N	number of tasks (components to be assembled).
M	number of resource types (robots, pallets, etc).
i	task index, $1\leq i\leq N$.
j	resource index, $1\leq j\leq M$.
k	scheduling decision index, $1\leq k\leq N$.
T	dispatching time.
$P(i)$	duration of task i.
$C(i)$	ending time of task i.
$\Gamma(i)$	set of precedence tasks of task i.
$R(i,j)$	number of units of resource j required by task i.

$R(j)$ number of available units of resource j (initialized as $R_0(j)$).
X set of tasks to be scheduled (initialized as X_0).
Y set of tasks satisfying the precedence constraints ($Y \subseteq X$)
Z set of tasks satisfying the precedence constraints and resources availability ($Z \subseteq Y$).
z^* first task of Z
W set of tasks being executed.
S set of tasks being executed with closer ending time.
$\rho_{[k]}$ rule of the scheduling decision k in vector of rules r.
C maximum ending time of the scheduled tasks.
$C_{\max}$ time needed to solve all the tasks (makespan).

Begin A1

1. Initialize:
   $T \leftarrow 0$
   $k \leftarrow 1$
   $C(i) \leftarrow \infty \quad \forall i\ (1 \le i \le N)$
   $R(j) \leftarrow R_0(j) \quad \forall j\ (1 \le j \le M)$
   $X \leftarrow X_0$
   $W \leftarrow \emptyset$
2. Create Y:
   $Y = \{y \in X : (C(x) \le T \ \ \forall x \in \Gamma(y)) \vee \Gamma(y) = \emptyset \}$.
3. Create Z:
   $Z = \{z \in Y : R(z,j) \le R(j)\ \forall j\}$
   IF $Z = \emptyset$ GO TO 6
4. Schedule the task:
   Arrange Z according to rule $\rho_{[k]}$
   $C(z^*) = T + P(z^*)$
   $R(j) \leftarrow R(j) - R(z^*, j)\ \forall j$
   $W \leftarrow W + \{z^*\}$
   $C = \max[C(w)]$ with $w \in W$
   $Y \leftarrow Y - \{z^*\}$
   $X \leftarrow X - \{z^*\}$
   IF $X = \emptyset$ GO TO 7
5. Increment the decision pointer:
   $k \leftarrow k+1$
   GO TO 3
6. Release of resources:
   Search $S = \{ s : C(s) = \min[C(w)]$ with $w \in W \}$
   $R(j) \leftarrow R(j) + \Sigma_{s \in S} R(s,j)\ \forall j$
   $T \leftarrow T + C(s)$ with $s \in S$
   $W \leftarrow W - S$
   GO TO 2
7. Determine $C_{\max}$:
   $C_{\max} = \max[C(w)]$ with $w \in W$

End A1

The application of the algorithm **A1** to the assembly example previously described in Section 3 with the vector of rules r = (40,40, 7, 7, 7,40,40,40,40,40,40,40) gives the solution 1 shown in Figure 2. It is interesting to remark that using rules #7 and #40 independently the obtained solutions last 62 and 63 seconds respectively.

In general, given a vector of rules r and an algorithm A it is possible to define a heuristic h from the pair (r, A), i.e. $h = h(r, A)$.

The algorithm **A1** can be used for any local search procedure that allows the generation of neighbor solutions in the heuristics space. Then, the rules are altered instead of the tasks.

5 The Genetic Algorithm

The generation of solutions in the heuristics space (i.e. vectors $r = (\rho_{[1]}, ..., \rho_{[k]}, ..., \rho_{[N]})$ to be used by **A1)**, was done using the genetic algorithm **GA1** described below.

Nomenclature:

I number of individuals (vectors of rules) in the population.
L number of iterations (generations).
p instance of the problem to be solved.
Π_r population of ancestors of the sequences of rules.
Π_h population of ancestors of the heuristics.
Π_s population of ancestors of the solutions.
Δ_r population of descendants of the sequences of rules.
Δ_h population of descendants of the heuristics.
Δ_s population of descendants of the solutions.
Λ_r population of mutated descendants of the sequences of rules.
Λ_h population of mutated descendants of the heuristics.
Λ_s population of mutated descendants of the solutions.
Ω_r population of eligible sequences of rules for the next iteration (generation).
r_i element i of the sets Π_r , Δ_r , Λ_r and Ω_r.
h_i element i of the sets Π_h , Δ_h , Λ_h and Ω_h.
s_i element i of the sets Π_s , Δ_s , Λ_s and Ω_s.

Begin GA1

Phase A: Initialization

0. Generation of initial populations:
   0.1 Generate the initial Π_r of I as:
   $\Pi_r = \{ r_i = (\rho_{[1]}, ..., \rho_{[N]}) : \rho_{[1]} = ... = \rho_{[N]} \}$
   0.2 Generate the initial population of heuristics:
   $\Pi_h = \{ h_i = h_i(r_i, \mathbf{A1}) : r_i \in \Pi_r \}$
   0.3 Generate the population of solutions of p and evaluate their makespan:
   $\Pi_s = \{ s_i = h_i(p) : h_i \in \Pi_h \}$
   0.4 Save as heuristic and incumbent solution the pair (h^*, s^*) with the best makespan.
   0.5 Determine the fitness f_j of the elements of Π_s as:

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

with:

D_i makespan of the solution i

$D_{\max}$ greatest makespan of the population

$D_{\min}$ lowest makespan of the population

α index of the population homogeneity

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

Phase B: Iterate through the following steps L times:

1. Selection of ancestors:
   Build $I/2$ pairs of elements of Π_r according to the fitness of the elements of Π_s.
2. Choice of the pairs for the crossover:
   - 2.1 Determine the probability of the current crossover: $P_c = P_c(\alpha)$.
   - 2.2 Assign a random number to each pair of sequences of rules.
   - 2.3 Decide, for each pair of sequences of rules, if a crossover should be done according to their random number and P_c.
3. Generation of descendants:
   - 3.1 Crossover the selected pair of sequences of rules to generate two descendants, creating Δ_r.
   - 3.2 Generate Δ_h and Δ_s from Δ_r as it was done in 0.2 and 0.3 respectively.
   - 3.3 Determine the makespan of the elements of Δ_s If any element of Δ_s has a better makespan than the incumbent solution, then save as heuristic and incumbent solution the pair (h^*,s^*) associated to that element.
4. Mutation of descendants:
   - 4.1 Determine the probability of mutation of the current generation: $P_m = P_m(\alpha)$.
   - 4.2 Assign a random number to each element Δ_r.
   - 4.3 Decide the elements of Δ_r to be mutated according to their random number and P_m.
   - 4.4 Mutate the chosen elements of Δ_r creating Λ_r.
   - 4.5 Generate Λ_h and Λ_s from Λ_r as it was done in 0.2 and 0.3 respectively.
   - 4.6 Determine the makespan of the elements of Λ_s. If any element of Λ_s has a better makespan than the incumbent solution, then save as heuristic and incumbent solution the pair (h^*,s^*) associated to that element.
5. Regeneration of the population:
   - 5.1 Build the population of eligible elements $\Omega_r \leftarrow \Pi_r + \Delta_r + \Lambda_r$
   - 5.2 Determine the fitness of the elements of the populations Δ_s and Λ_s as it was indicated in 0.5.
   - 5.3 Choose I elements from Ω_r according to the fitness of the elements of Π_s, Δ_s and Λ_s.

End GA1

6 Experimental Results

In order to validate the proposed approach, 270 different instances of the task sequencing problem have been solved considering:

Number of tasks: $6 < N < 15$.
Types of resources (robots, pallets, etc): $1 \le M \le 3$.
Units of resource j (e.g. number of robots) $2 \le R_o(\mathrm{j}) \le 5$.
Duration of task i: $1 \le P(i) \le 16$.
6 different ratios #precedence/N.

The proposed **GA1** has obtained the optimum solutions of the 270 instances of the problem in less than 12 minutes using a Pentium II 233MHz.

The following subsections detail some particular aspects of the implemented algorithm.

6.1 Initial Population

A population with size I =100 was used with the aim of considering all the heuristics derived from the rules shown in Appendix A. Then, the initial population is composed of 100 vectors of rules, each one composed by one particular rule in all its components.

This allows the exploration of all the solutions generated by the greedy heuristics. In order to increase the size of the initial population it is only necessary to include new rules or generate hybrid rules using linear combinations of previous rules.

The random selection of the task to be dispatched has also been incorporated as a rule (rule 26), to allow the generation of any solution. The rule including random selection is necessary when the set of rules does not guarantee the generation of all the solution space.

6.2 Selection Process

The elements of Π_r are randomly selected with larger probabilities for those elements with better fitness.

6.3 Probabilities of Crossover and Mutation

The probability of any crossover or mutation, P_c and P_m respectively, depends on the homogeneity index α of the current population. In this way, a population with quite similar individuals will be modified through mutations because the crossovers would not be effective for diversification. The following values have been used for the experiments: P_c =1-0.5α, and P_m = 0.05+0.95α.

6.4 Crossover Process

Given two vectors of rules (the ancestors) two components are randomly selected, and the rules between these components in both of them are interchanged to obtain two new vectors of rules (the descendants). Figure 3 illustrates a crossover.

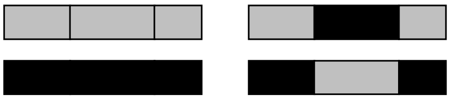

Figure 3: Example of a crossover.

6.5 Mutation Process

Three different types of mutation of a vector of rules have been considered: soft, medium and hard. The probability of each type of mutation is given a priori.

Soft mutation: two components of the vector of rules are randomly selected and interchanged.

Medium mutation: one rule of the vector is randomly selected, this rule is successively replaced by all the available rules and the combination that generates the lowest makespan is selected.

Hard mutation: it is equivalent to test all the possible soft mutations of the vector of rules and select the one with the lowest makespan.

6.6 Regeneration Process

The elements of Ω_r are randomly selected giving higher priority to those sequences of rules with better fitness.

6.7 Frequency of the Rules

The frequency of the 100 rules in the 270 optimum solutions was also determined (Figure 4). The initial frequency of each rule is 1%, but the results show that, after the evolutionary process, some rules appear with higher frequency; in particular, it is interesting to note the 6% of rule 26 (random selection of a task).

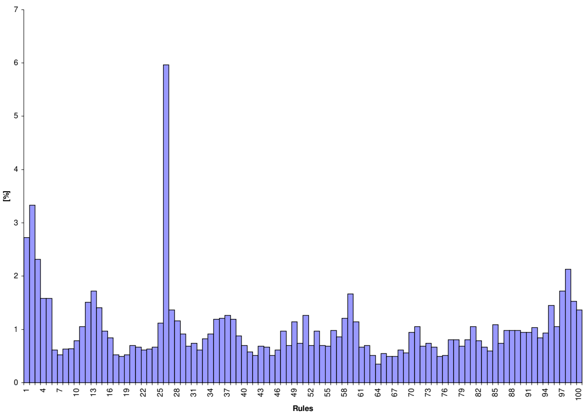

Figure 4: Final frequency of the 100 rules in the experiments.

7 Conclusions

A method to look for solutions of the RCPSP with application to the scheduling of assembly operations with limited resources (the robots) has been presented. The main contribution of the method is the incorporation of the knowledge provided by the specific heuristics of the problem in a local search procedure. In this way, the solution is characterized by a sequence of priority rules. The method has been implemented using a genetic algorithm. The experiments and computational experiences were quite satisfactory.

Appendix A: List of Rules

Nomenclature:

$P(i)$	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.

Schedule the task z^* : $v(z^*)=\max_{i\in Z}[v(i)]$

NAME	RULE:
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	$v_{30}(i) = -\sum_{h \in Z - \{i\}} max[0, (EFT(i) - LST(h))]$

*A different rule is considered for each value of w_r such that $w_r \in \{0, 0.1,...,0.9,1\}$, w_p =1- w_r .

References

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

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

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

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

[5] Simpson, W.P. & Patterson, J.H. (1996): "A multiple-tree search procedure for the resource-constrained project scheduling problem". *European Journal of Operational Research*, 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] Holland, J.H. (1975): *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Mich.

[11] Goldberg, D.E. (1989): *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, Mass.

[12] Padman, R. & Roehrig, S.F. (1997): "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*, 405-421. R.S. Barr & R.V. Helgason & J.L. Kennington (Ed.). Kluwer, Norwell, MA, USA.

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

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

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

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