HEURISTICS AND HIPERHEURISTICS FOR SEQUENCING MIXED MODEL ASSEMBLY LINES MINIMIZING WORK OVERLOAD

Joaquín Bautista Valhondo
NISSAN Chair UPC, – Escuela Técnica Superior de Ingeniería Industrial de Barcelona – Universitat Politècnica de Catalunya, Barcelona, Spain
joaquin.bautista@upc.edu

Jaime Cano Belmán
Departament d'Estadística i Investigació Operativa – Universitat Politècnica de Catalunya
Barcelona, Spain
jaime.cano-belman@upc.edu

Abstract – There are different approaches for the mixed model sequencing problem on assembly lines. In this paper the goal of minimizing work overload is treated. Since solve this problem optimally is difficult, we test constructive procedures, local search and a new hiperheuristic procedure.

Keywords: Sequencing, assembly line, work overload, local search, priority rules, JIT.

I. INTRODUCTION

Assembly lines are commonly used in automotive industry. Two important decisions for managing mixed model assembly lines are to spread out work in stations (balancing) and determine the sequence to introduce cars in to the assembly line (sequencing). When the medium term decision of balancing has been taken, sequencing decision must be considered. Depending on the manufacturing environment it may be desirable minimize or maximize some parameters or characteristics of the line [1]. One of the two main criteria [2] for sequencing mixed models on assembly lines considers the labelling of load on stations. The problem addressed in this paper considers the objective of minimizing the work overload.

Since a processing time for a product in a station can be grater than cycle time, there is a maximum quantity of those products that can be consecutively introduced in the line without causing a delay in the finishing of works. All those jobs with high and low work content must be under control for avoiding excessive work load and idle time. Stations are confined by upstream limit and down stream limit. That implies that products are mounted on an assembly line that moves at constant speed and workers can do their job on products only when they are inside their station. Products get in the station at constant time intervals. Work overload occurs when work on a product can not be finished before it leaves the station.

Initial works on this criterion were carried out by [3] or [4]. In this paper an extension of a procedure in [3] is proposed, which considers not only two different jobs/products (basic product and special product, differentiable by their poor and rich work content respectively), but also multiple products. Other literature related with the problem is [5], [6], [7] and recently [8].

Inspired on procedures from [4] and [9], four procedures are proposed, which also consider multiple products and multiple stations. We use local search with different neighborhoods for improving the solutions obtained with constructive procedures. A new hiperheuristic is tested with the sequencing problem.

This paper is organized as follows: section 2 contain our constructive proposals, in section 3 we apply local search, section 4 contains a proposal of a new hiperheuristic procedure, section 5 shows computational experience for constructive procedures, local search and the hiperheuristic. In section 6 conclusions are mentioned.

II. WORK OVERLOAD MEASUREMENT

In [3] a general formulation for measuring work overload is proposed. Work overload is measured in time units and the time unit is the cycle time c (time between product arrivals into the station). Let L denote the station length (or time window), p_{ik} the processing time for the job on the product i (i=1,…,I) in the station k (k=1,…,K), s_t the starting instant of the job in the position t ($s_1=0$; $s_t=max(t\text{-}1,f_{t\text{-}1})$), and f_t the finishing instant of the job in position t ($f_t=min(s_t+p_i,t\text{-}+L)$). Given a sequence of size T (t=1,…,T), and considering only one station, wo_t is the work overload obtained in position t of the sequence: $wo_t = [p_i+s_t-(t\text{-}1+L)]^+$ where $[x]^+ = max(0,x)$. The total work overload is $z = \sum_t wo_t$. Mathematical programming formulations of the problem can be found in [3] or [6]. The problem is difficult to solve due to the lack of structural properties. The problem has a lot of possible solutions and a big effort to evaluating those solutions is required. As explained in [3], [6] or [8] the problem is considered to be NP-hard.

To elucidate the reader on the work overload problem, a single station illustrative example is shown. Four products are considered: A, B, C, and D, with the following processing times (0.82, 0.94, 1.19, 1.15), and demands (3,5,7,1). Station length L=1.2. Processing times and station length are expressed in cycle time units (c=1). Let us assume products are going to be introduced into the assembly line in the following order: CCCADCCBCABABCBB.

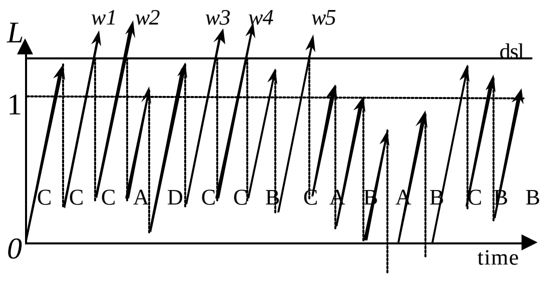

Figure 1- Worker movement diagram

Without loss of generality, the initial worker position is assumed to be on the upstream limit of the station. Figure 1 represents the movement of the worker during his job on the products according to the sequence established above. Arrows represent the processing times, and dotted lines represent the worker displacement from one finished product to the next product on the line. Station length is limited by the down stream limit (*dsl*). The first job is done on a product kind C, which requires 1.19 time units. When this job is finished, the worker walks upstream for reaching the product C in position 2 of the sequence. We assume this time is negligible because the velocity of the worker is greater than the conveyor speed. Then, the worker starts the second job 0.19 units away from the upstream station limit and would finish it 0.19+1.19 units away from the upstream limit. Nevertheless, the worker can not go beyond the *dsl*, and 0.18 units of work must be left unfinished ($w1$). When the worker reaches the *dsl*, he leaves his job and walks upstream, and starts working on the product in position 3 of the sequence 0.20 units away from the upstream limit. Again, the time allocated is not enough to finish the job, and 0.19 units of work overload are produced ($w2$). The product in position 4 requires 0.84 time units, and the work on it is finished when the worker is 0.20+0.84 units away from the upstream limit. This time, the job is completed. Products in positions 6, 7 and 9, also produce work overload ($w3$=0.16, $w4$=0.19, $w5$=0.13). The total work overload produced by the sequence is 0.85 cycle time units.

III. Constructive Procedures

In this section, four procedures are proposed inspired on the works from [3], [4] y [9]. Our proposals assume multiple products and only one station. Then those one station procedures are used for determining work overload prediction values in a multi station procedure taken from [3]. An original procedure (Y&R) from literature is described to make more understandable the proposed extension.

A. Y&R Procedure

The original procedure from [3] considers single station and two kinds of products: products with optional components and basic products. Let A denote special products, which have processing time greater that c, and B denote basic products with processing time smaller than c. The procedure is based on the repetition of a stable subsequence composed by m_a units A, and m_b units B. Since processing times for special products p_a is greater than c, there is a limit on the number of units A that can be consecutively sequenced, without causing work overload. This limit (X) is the maximum integer satisfying $X = (L-c)/(p_a-c)$, and is also the maximum value m_a can take. The procedure tries to regenerate, that is, to bring the worker back to the beginning of the window (its assumed original position), after a cycle composed by m_a units A and m_b units B.

The subsequence of m_a and m_b units, can be determined with the equation $m_a{\cdot}p_a+m_b{\cdot}p_b=m_a+m_b$, where $m_a \le X$. Then, assuming there are integer values for m_a and m_b, maximum utilization is achieved by solving the next nonlinear MP model (1).

$$\begin{aligned} &\text{maximize} \quad (p_a \cdot m_a + p_b \cdot m_b)/(m_a + m_b) && (1)\\ &\text{s.t.} \quad m_a \le X \\ &\qquad p_a \cdot m_a + p_b \cdot m_b = m_a + m_b \\ &\qquad m_a, m_b \ge 0, \text{ integers} \end{aligned}$$

Let n_i the quantity to be produced of product i ($T=\sum n_i$), and nc the maximum number of cycles composed by m_a units A, and m_b units B. The sequence is built according to the following steps: 1) assign the nc cycles, 2) assign $x_a=\min(n_a-nc{\cdot}m_a,\ m_a)$ products A, 3) assign $x_b=n_b-nc{\cdot}m_b$ products B, and 4) if necessary assign $n_a-nc{\cdot}m_a-x_a$ products A.

B. Y&R Procedure Extension

This extension considers not only two but also multiple products. As in the original procedure, our extension (YRx) builds sub-sequences composed by m_i units of the product i. While there are enough unscheduled units, the sub-sequence is repeated. Otherwise, a new subsequence is calculated with the remaining products. Let **A** the set of products with processing time greater than the cycle time (p_i>1), **B** the set of products with processing time smaller than the cycle time ($p_i \le 1$), and x_i the maximum consecutive quantity of product i that can be scheduled without causing overload or idle time.

$$\begin{aligned} &\text{maximize} \ \sum_{i=1}^{I} p_i \cdot m_i \Big/ \sum_{i=1}^{I} m_i && (2)\\ &\text{s.t.} \ m_i \le \min\{x_i, d_i\} \\ &\qquad \sum_{i=1}^{I} p_i \cdot m_i \le \sum_{i=1}^{I} m_i \\ &\qquad m_i \ge 0, \ \text{integers} \end{aligned}$$

Model (2) is the extension proposal of the original model (1), which finds the quantity of each product contained in the sub-sequence.

In model (2), $x_i \leq L\text{-}c/p_i\text{-}c$, $\forall i \in \mathbf{A}$, and $x_i \leq L\text{-}c/c\text{-}p_i$, $\forall i \in \mathbf{B}$. d_i represents the remaining production of product i. When t=0, $d_i = n_i$. To arrange products in the subsequence, we consider alternatively the set **A**, and then the set **B**. Let c denote the cycle time. The order in which units will be incorporated into the sequence is done according the diminishing value of index $r_i = m_i \cdot |c\text{-}p_i|$. Thus, from the set **A**, the item with the bigger index r_i (let j) is selected and the m_j products j are consecutively assigned. Then, from the set **B**, the item with the bigger index r_i is selected (let l), and the m_l corresponding products l are assigned; and so on. Since it is easier to solve a linear model than a non-linear model, we have transformed the non-linear model (1) into a linear one. Then we used the transformed model for finding subsequences.

C. Greedy Procedures

The proposed procedures try to favour the movement of the worker in the station from the lower to the upper limit of its station (*Up-Down*) as depicted in figure 2. Arrows represent the processing times, and dotted lines represent the worker displacement from one finished product to the next product on the line. If a long arrow is scheduled after the unit has been completed (in the circle of figure 2), work overload is produced, then, *down* step is required to avoid it.

With those two *up-down* cyclic steps, we attempt to regenerate. Perfect regeneration is reached only for certain parameters values [3].

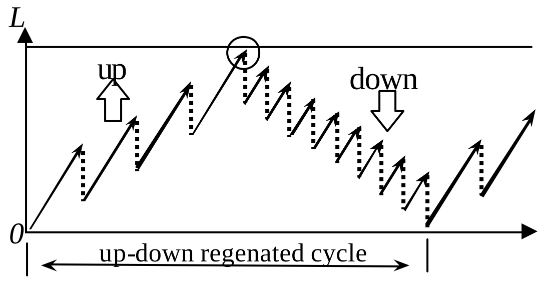

Figure 2- Up-down worker movement diagram

During the *up* step, only products with processing time greater than c are considered. Products with processing time smaller than c are considered during the *down* step.

For deciding the kind of product to sequence in the period t, we make use of a dynamic index $r_i = d_i \cdot |c\text{-}p_i|$, where i belongs to the set **A** or **B** considered in the current stage t. If t=0, d_i=n_i. Since r_i depends on the pending production of product i, it must be updated in each phase. The product of the set under analysis with the bigger index r_i is selected to be assigned in period t. s_1=0 is assumed.

Following those ideas, three procedures are proposed: Ud, UdC and UdR. Ud allows commit in idle time only ones in each cycle reaching the complete regeneration; while it is possible; UdC forbids incurring in either idle time or work overload, putting them off; UdR allows a maximum *quantity* of work overload until the position t of the sequence. This *quantity* is limited by lbw/T^*t. lbw is a lower bound of work overload considering multiple stations.

$$lbw = \sum_{k=1}^{K} \left[\sum_{i=1}^{I} n_i \cdot p_{ik} - \left(c \cdot (T-1) + L_k\right) \right]^+ \qquad (3)$$

Where $[x]^+$ is max(0,x). UdR tries to distribute or regularize work load along the sequence.

D. Multi-station procedure

A multi-station procedure is taken from [3]. This procedure is based on single station procedures. Single station procedures are used for determining work overload predictions. The sequence is built progressively in such a way that for each sequencing instant t, the product with remaining production and the best overload predictor is selected and assigned in the t^{th} position of the sequence.

Let: wp_{ik} be the work overload predictor in station k due to the assignment of product i, $w_i(k,t)$ be the work overload obtained in station k in period t of the sequence due to the assignment of product i, s_{kt} be the starting instant of job in station k in period t (or initial worker position in station k in period t), and $sc_i(t)$ be the total work overload prediction produced by product i in period t.

```
0. Initialize $s$=0.
**for** ( $t$ =1 **to** $T$ )
    **for** ( $i$=1 **to** $I$ )
        1. Assume $sequence(t) \leftarrow i, (d_i\text{-}1)$.
        2. Compute predictor $sc_i(t)$ for each station.
        3. $d_i$+1.
    **end for**
    4. $sequence(t) \leftarrow i^* : sc_{i^*}(t) = \min_{i \in I} \{sc_i(t)\}$.
    5. Update data.
**end for**
```

wp_{ik} is obtained with single station procedures. The work overload produced in period t due to the assignment of product i is obtained by (5) and (6).

$$sc_i(t) = \sum_{k=1}^{K} \left(wp_{ik} + w_i(k,t)\right) \cdot sc_k \qquad (4)$$

$$w_i(k,t) = [s_{kt} + p_{ik} - ((t\text{-}1)^{\cdot}c) - L_k]^+ \qquad \forall\, i \in \mathcal{A} \qquad (5)$$

$$w_i(k,t) = [(t^{\cdot}c) - s_{kt} - p_{ik}]^+ \qquad \forall\, i \in \mathcal{B} \qquad (6)$$

IV. Local Search

For improving the solutions obtained with the four constructive procedures described in the previous section, local search is applied. Two well known kinds of neighborhoods had been used in the search: swap and insertion.

A. Swaps

Swaps of two and three elements of the sequence had been considered. Swap of two elements (2S) is simple. One solution can just produce a new one. Nevertheless, swap of three elements have five possible neighbors solutions. From those five possibilities, only in two of them the three elements considered take a new position in the sequence: (b,c,a) and (c,a,b) in figure 3.

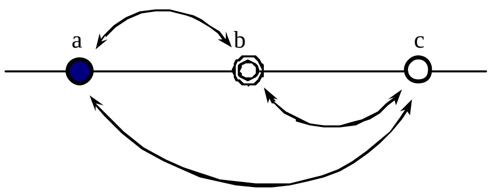

Figure 3- 3 swap neighbors

Then, we had used two different 3-swap neighborhoods: 3S(a) and 3S(b). 3S(a) considers the tow changes where all of the elements considered take a new position in the sequence. In the other hand, 3S(b) take into account the five possibilities. We had also considered the idea of applying 3 swap after no improvement can be found with 2 swap, we call that 2-3S(b).

B. Insertion

By the insertion, a new neighbor from the current feasible sequence is obtained getting a segment of certain size from the solution, and then it is inserted in a different position of the sequence. Even tough size of segments (let *or*) can take the value $1 \leq or \leq T$, in the computational experience only had been tested $2 \leq or \leq 10$. The size of the Insertion neighborhood is smaller than the Swap neighborhood; therefore, the computational effort is smaller too.

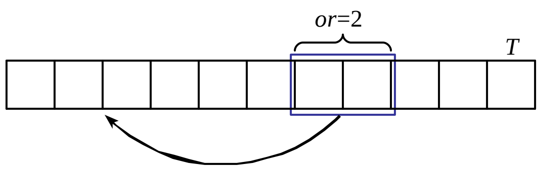

Figure 4- Insertion neighbors

In all the local search experience, the maximum number if iterations with out improvement has been established to $\sqrt{T}$.

V. Priority Rules

In this section a hiperheuristic is described. The procedure is inspired on the Scatter Search (SS) Meta heuristic. Instead of using feasible solutions for producing new ones, our proposal use priority rules chains. The objective value of a chain is obtained getting the corresponding solution sequence. That is done using a constructive procedure of rules combination (PCCR), and measuring its work overload value.

A. PCCR

PCCR is a greedy constructive procedure based in the combination of priority rules. The assignation of a product in certain position of a sequence depends on the priority rules. A set of rules $R=\{r_1,r_2,r_3,...,r_R\}$ establishes the order of the products in a sequence. The chain (sequence of priority rules) will have the same number of rules as positions have a product solution sequence (T). The rule in the position t of the chain, determines from a set of products, the product i that best satisfies the rule r.

Given a rules chain size T, the determination of the product that best satisfies the rule r of the position t of the chain is obtained as follows:

- For each candidate product, compute the value of the application of the rule r.
- Select the product i with the best value for the rule r.
- Assign the product i in the position t of the products sequence.
- Update pending demand for product i.

B. Hiperheuristic

Similarly to SS [10], our proposal is an evolutionary algorithm that creates new elements combining the existing ones, improving this way the criterion used to evaluate the elements. Our proposal operates on a Reference Set (*RefSet*). But, instead of a reference set of solutions, we use a reference set of rules chains. Combining those rules chains, new rules chains are created. A typical *RefSet* size in SS is 20 or less, while the size of our *RefSet* is in function of the number of rules R considered.

The following is a pseudo code of the proposed procedure:

```
start
    0.1 Create RefSet static and dynamic.
    0.2 Initialize frequency matrix, Fr.
while ( Diversifications < Max Diversifications )
    1. Combine rules of the RefSet.
    2. Regenerate RefSe.t
    if ( RefSet state is not improved )
        3. Diversify RefSet.
    end if
end while
```

The *RefSet* is conformed by two tiers: the static subset (*RSs*) and the dynamic (operative) subset (*RSd*). The size of both *RSs* and *RSd* is R.

RSs is called “static”, because it is not modified during the search process. In *RSs* the rules chain 1 contains only the rule 1, the rules chain 2 contains only the rule 2, and so on. In this way the procedure ensures the consideration of all the priority rules in the combination phase.

$$cr_r = (r,r,r,...,r) \rightarrow RSs = \{cr_r : r \in R\} \qquad (7)$$

The dynamic *RefSet* changes in each iteration of the search process. The rules chains of the initial *RSd* are generated randomly. *RSd* contains the elite group of rules chains. *RSd* is updated in each iteration of the process taking into account the new chains with the best values of work overload obtained by the last combination.

$$1 \le RSd_{it} \le R \qquad (8)$$

The work overload value for a chain is obtained applying the PCCR. In this work, 20 priority rules had been used. If during the PCCR procedure, one or more products have the best value for the rule in period *t* of the sequence, the tie is eliminated taking only the products in the tie and applying the rules in order (starting with rule 1), until the tie disappears.

Different criteria are considered in the priority rules used in the procedure. Rules 1-4 decide which product is going to be assigned using the processing times data. Rules 5 and 6 select the product with the bigger and smaller pending demand respectively. Rules 7,8,14 and 15 differentiate the products making a relation of pending production and the difference between the processing time and cycle time. The displacement of the workers in the stations is used for selecting the product in rules 9 and 10. Bottleneck station processing times are considered in rules 11 and 12. Rules 16 and 18 use the work overload caused by the assignment of a product. Rules 17 and 19 use idle time. Rule 13 select a product using the measurement of the regularization of the load along the sequence. Similarly, rule 20 select according the regularization of idle time.

In each iteration of the process, a frequency matrix is obtained *Fr*(*r*,*t*) , which contains the number of times that a rule *r* is in the position *t* of the chains in the *RSd*. Since *RSs* do not change, it is not necessary consider it for computing *Fr*. The *Fr* matrix is used in the combination of the rules chains in the *RefSet*. *Fr* is also used in the diversification phase.

When two rules chains (*parents*) are combined, a new one (*son*) is obtained.

The element in the position *t* of the son chain is determined according to the frequency that the rule *r* has in the position *t* in the *Fr* matrix.

$$cr(t) = \begin{cases} cp(t) & \text{if} \quad cp(t) = cq(t) \\ cq(t) & \text{if} \quad Fr(cp(t),t) \ge Fr(cq(t),t) \\ cp(t) & \text{otherwise} \end{cases}$$

Once all the *son* chains had been obtained, the PCCR is used for getting the work overload value for the new chains. Those with the best values are considered for updating the *RSd*. That is, the *RSd* is *regenerated*.

Three regeneration alternatives are analyzed in this paper:

- The *RefSet* is regenerated with the best chains, considering both, the parents set and the *sons* set.
- The RefSet is regenerated with the *R* best chains in the *sons* set.
- The worst *aR* chains in the *RefSet* are regenerated by the *aR* best chains in the *parents* set.

In the regeneration process, duplication must be avoided. Then, all the chains in the *RefSet* have different work overload values.

When the regeneration process does not produce improvements, the *RefSet* must be diversified. Diversification is done in two steps: 1) creation of diversified chains, and 2) selection of those diversified chains which are the least similar to each other.

Step 1 of diversification is done using the information contained in the frequency matrix *Fr*. When diversification is necessary, the combination of rules is done in a different way. Given two *parent* chains *p* and *q*, one diversified *son* chain is obtained. The difference in the way chains are combined in the diversification phase is the following: in position *t* of the new diversified *son* chain, the rule of the parent chain that has the *smaller* value in the frequency matrix *Fr* is assigned. The idea is to create chains containing rules with inferior frequency in the *Fr*, so the space of solutions explored is changed.

The second step of the diversification process iteratively looks for diversified chains that are different with respect to the chains inside the current *RSd*. The grade of differentiation between two chains is measured with the number of coincidences. A coincidence exists if in the same position *t*, both of the chains have the same rule *r*.

VI. Computational Experience

The proposed procedures are tested with the battery of problems designed by [6]. In all the instances *c*=90 time units. Instances do not consider weight (cost) by incurring in work overload or idle time in stations. For measuring the quality of the solutions obtained by the proposed procedures we use the same global index used by the battery designers.

Originally, instead of weak lower bounds, the objective function values wo^*_h of the best known solution for an instance *h* is used when comparing procedures. Since this measure is not defined for $wo^*_h = 0$, the following aggregated relative deviation is used:

$$rel.wo : \left(\sum_{h=1}^{100} wo_h - \sum_{h=1}^{100} wo^*_h\right) \Big/ \sum_{h=1}^{100} wo^*_h \cdot 100\% \qquad (9)$$

We have added two more indexes for aggregated relative deviation. The original index (9) is represented by *rel.wo*2. In *rel.wo*1 the value wo^*_h is the lower bound obtained by (3). In *rel.wo*3 the value wo^*_h is the best solution founded considering the results obtained with CPLEX after 15 minutes of search. Computations are performed in a Pentium 4 CPU 2.4 GHz, 512 MB RAM under a system Microsoft windows XP professional 2002. CPLEX 7.5 was used in YR-x procedure, and for searching the optimum during 15 minutes.

Table I shows the results for the three indexes evaluated for the four constructive procedures described in section III. The best indexes are reached by the procedure that tries to spread out the work overload along the sequence. In the other hand, our extension proposal for Y&R procedure produces not the expected results.

Table I- Global results for initial procedures

Index	Ud	UdC	UdR	YRx
rel.wo1	45.64	41.82	40.97	69.25
rel.wo2	5.49	2.73	2.11	27.90
rel.wo3	6.00	3.22	2.59	28.51
#best	19	42	44	2
Cpu	3.99	4.96	4.56	4276

Local search was also applied on the results of constructive procedures. Table II shows the value for the index rel.wo1 after LS. 2S, 3S(a), 3S(b) and 2-3S(b) correspond to Swap neighborhoods. 2-10 Ins are segment insertion neighborhoods with segment size from 2 until 10. In the search by three elements sweeping, the computational effort is much bigger than the effort required in 2 elements sweep due to the size of the neighborhood. Applying 3 swap elements after 2 swap can improve the solutions.

Table II- Global results after LS, rel.wo1

*Rel.wo*1 (%)	Initial Procedure			
	Ud	UdC	UdR	YRx
Original	45.64	41.82	40.97	69.25
2S	24.32	24.30	24.53	24.54
3S(a)	35.16	33.03	34.84	55.39
3S(b)	32.99	31.86	30.55	44.15
2-3S(B)	23.96	23.88	24.09	24.06
2Ins	23.88	23.64	23.80	24.38
3Ins	23.51	23.35	23.34	24.22
4Ins	23.34	23.20	23.19	24.25
5Ins	23.15	23.00	23.00	24.02
6Ins	23.01	22.95	22.99	24.03
7Ins	22.98	22.87	22.77	23.91
8Ins	22.82	22.71	22.63	23.34
9Ins	22.72	22.61	22.60	23.35
10Ins	22.56	22.65	22.44	23.29

The neighborhood size when segment insertion is used, is much smaller, and requires less computational effort than swapping. Furthermore, according to global indexes, work overload solutions are better. Cpu time limit was imposed in the search of local optimums. The limit for LS with Swaps was 3600 seconds, except for 2-3S(b), which had 3600 for the 2Swap search, and 1800 seconds for the 3S(b) search. In segment insertion search, time limit was 1800 seconds.

Rel.wo2 index is shown in table III. Since better solutions had been founded during LS, the original rel.wo2 value for some initial procedures is different to those values in table I. In general, better results are obtained when LS is applied on the UdR initial procedure, and for our problem, segment insertion neighborhood overwhelms swapping neighborhoods.

Table III- Global results after LS, rel.wo2

*rel.wo*2 (%)	Initial Procedure			
	Ud	UdC	UdR	YRx
Original	20.12	16.97	16.37	40.16
2S	2.53	2.51	2.70	2.71
3S(a)	11.47	9.72	11.1	28.16
3S(b)	9.68	8.75	7.67	18.89
2-3S(B)	2.23	2.17	2.34	2.32
2Ins	2.17	1.97	2.11	2.58
3Ins	1.86	1.74	1.72	2.45
4Ins	1.73	1.61	1.60	2.48
5Ins	1.57	1.45	1.44	2.28
6Ins	1.45	1.40	1.44	2.29
7Ins	1.43	1.34	1.26	2.20
8Ins	1.29	1.21	1.14	1.73
9Ins	1.21	1.12	1.12	1.73
10Ins	1.08	1.16	0.98	1.69

After LS, all results are better than those obtained 15 minutes of search using the optimization software CPLEX 7.5, then, the index rel.wo3 loss worth.

Table IV- Results for HH1, rel.wo1

*rel.wo*1 (%)	Regeneration type		
Max Diversif.	Reg1	Reg2	Reg3
D3	42.81	42.79	42.54
D4	42.72	42.74	42.45
D5	42.64	42.64	42.47
D6	42.53	42.55	42.33

A computational experience has also been done for the hiperheuristic proposed procedure. This experience is not related with the solutions values from previous experiences. Our analysis considers the combination of three kind of regeneration used in the procedure and the maximum number of diversifications (first column of tables).

Table V- Results for HH1, rel.wo3

*rel.wo*3 (%)	Regeneration type		
Max Diversif.	Reg1	Reg2	Reg3
D3	1.85	1.84	1.66
D4	1.78	1.80	1.60
D5	1.73	1.73	1.54
D6	1.65	1.66	1.51

Table IV shows the global results for the index rel.wo1. The performance of the three regeneration methods is similar. As can be expected, when more diversifications are applied, results are improved; nevertheless, this improvement is small. Index rel.wo3, considers also the CPLEX result after 15 minutes of search. Work overload solutions are almost as good as the solutions obtained considering CPLEX. In average, 156 seconds are needed by the procedure to finishing the search.

VII. CONCLUSIONS

This work treats with a variant of the problem of sequencing products (mixed models) on a paced assembly line. We consider the approach in which a product demands a component (attribute), which has different versions, and requires different processing times in the application of each. The aim of these procedures is to minimize work overload (lost work) in all the stations of the assembly line due to the limited time spared in the stations and to the work loads along a given sequence. Both boundaries of stations are closed, and we assume as in [3], and [6], the displacement time the worker need to go from one product in to the next, is negligible. We compiled some procedures founded in literature: [3], [4] and [9], and we propose greedy procedures inspired on the previous procedures. Procedures consider more than two different products and multi stations.

Results obtained for constructive procedures dealing with the deviation in relation to a bound, are highly satisfactory, results also show CPU time required for the greedy proposed procedures is acceptable for big instances, and when work overload is spread our along the sequence, better results are obtained.

When LS is applied on the constructive procedures results, index rel.wo1 is improved to almost the half. Local search with segment insertion outperforms the elements swap neighborhood, requiring less computational effort.

The hiperheuristic proposed procedure get good results which are comparable with those obtained with the constructive procedures.

REFERENCES

[1] J. F. Bard, E. Dar-El, A. Shtub, “An analytic framework for sequencing mixed model assembly lines”, International Journal on Production Research, vol. 30,1, 35-48, 1992.

[2] Y. Monden, “Toyota production system”, Institute of Industrial Engineers Press, Norcross,GA, 1983.

[3] C. A. Yano, R. Rachamadugu, “Sequencing to minimize work overload in assembly lines with product options”, Management Science, vol. 37, 5, 572-586, 1991.

[4] A. Bolat, C. Yano, “Scheduling algorithms to minimize utility work at a single station on paced assembly line”, Production Planning and Control, vol. 3, 4, 393-405, 1992a.

[5] L Tsai, “Mixed-model sequencing to minimize utility work and the risk of conveyor stoppage. Management Science”, vol. 41, 3, 485-495, 1995.

[6] A. Scholl, R. Klein, W. Domschke, “Pattern based vocabulary building for effectively sequencing mixed-model assembly lines”, Journal of Heuristics, vol. 4, 4, 359-381, 1998.

[7] W. Zeramdini, H. Aigbedo, Y. Monden, “Bicriteria sequencing for just-in-time mixed-model assembly lines”, International Journal of Production Research, vol. 38, 15, 3451-3470, 2000.

[8] S. Kotani, T. Ito, K Ohno, “Sequencing problem for a mixed-model assembly line in the Toyota production system”. International Journal of Production Research, 42 (23), 4955-4974, 2004.

[9] A. Bolat, C. Yano, “A surrogate objective for utility work in paced assembly line”, Production Planning and Control, vol. 3, 4, 406-412, 1992b.

[10] M. Laguna, R. Martí, “Scatter search, methodology and implementations in C”, Kluwer Academic Publishers, USA, 2003