A Hyper-Heuristic to Sequencing Mixed-models on Assembly-lines Minimizing Work Overload[*]

Joaquín BAUTISTA[a], Jaime CANO[b],

[a] *NISSAN Chair, ETSEIB, Universitat Politècnica de Catalunya, Spain*

[b] *DEIO, Universitat Politècnica de Catalunya, Spain*

Abstract

There are different approaches for the mixed-model assembly-lines sequencing problem. In this paper the goal of minimizing work overload is treated. This approach considers the existence of time windows in each work station. Different versions of a product are considered to be assembled in the line (e.g. car industry), which require different processing time according to the work required in each work station. Long sequences of rich-work products can lead to produce work overload when stations cannot fulfil all the assigned tasks. Since solve this problem optimally is difficult, we test local search and a hyper-heuristic procedure. A computational experiment is used to detect the performance of the proposed procedures.

Key words: Hyper-heuristic, Scheduling, Work Overload, Local Search, Priority Rules.

1 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 to 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 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

[*] This paper was not presented at any other revue. Corresponding author: J. Bautista. Tel. +34 - 934011703.
Email addresses: joaquin.bautista@upc.edu, jaime.cano-belman@upc.edu.

problem is [5], [6], [7] and recently [8]. Inspired on procedures from [4] and [9], in [10] constructive procedures in proposed, which consider multiple products and multiple stations. We use local search with different neighbourhoods for improving the solutions obtained with constructive procedures from [10].

During the last years promising search algorithms called hyper-heuristics have been proposed to raising the level at which optimizations systems can operate [11]. The term describes the process of using (meta-)heuristics to chose (meta-)heuristics to solve the problem in hand. In theory, a hyper-heuristics should be cheaper to implement and easier to use than problem specific special propose methods and the goal is to produce good quality solutions in this more general framework. Some problems in which the hyper-heuristics performance has been testes are the timetabling [12] and bin-packing [13]. Then, looking for a whole search algorithms [14] a hyper-heuristics is proposed to solve the sequencing problem treated in this paper.

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 hyper-heuristic procedure, section 5 shows computational experience for constructive procedures, local search and the hyper-heuristic. In section 6 conclusions are mentioned.

2 Work overload

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-1,f_{t-1})), and f_t the finishing instant of the job in position t (f_t=min(s_t+p_i,t-1+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: $wo_t = [p_i+s_t-(t-1+L)]^+$ 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. The problem is considered to be NP-hard [3], [6] or [8].

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.

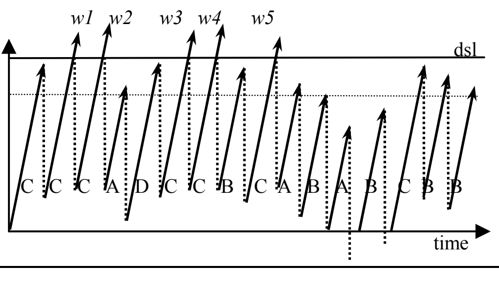

Fig. 1. Worker movement diagram

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 (*w*1). 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 (*w*2). 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 (*w*3=0.16, *w*4=0.19, *w*5=0.13). The total work overload produced by the sequence is 0.85 cycle time units.

3 Local search

Two well known kinds of neighbourhoods had been used in the local search (LS): swap and insertion. Four seed sequences are obtained with constructive procedures presented in [10].

3.1 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 neighbours 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). Three-swap is depicted in Figure 2.

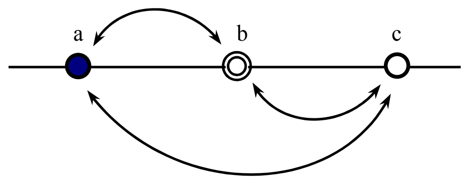

Fig. 2. Scheme for 3 swap

Then, we had used two different 3-swap neighbourhoods: 3S(a) and 3S(b). Neighbourhood 3S(a) considers the two 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 a local optimum has been found with 2-swap. This scaled search is references in this paper as 2-3S(b).

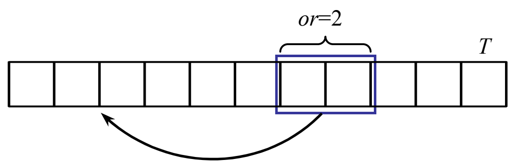

Fig. 3. Scheme for Insertion

3.2 Insertion

By the insertion, a new neighbour 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 $1 \le or \le T$, in the computational experience only had been tested $2 \le or \le 10$. The size of the insertion neighbourhood is smaller than the swap neighbourhood; therefore, the computational effort is smaller than the swap neighbourhood one. Nevertheless, to do complete neighbourhood search still requires a considerable computational effort. In all the local search experiments, the stop search criterion used is the maximum number of iterations with out improvement. This parameter has been established to $\sqrt{T}$. Figure 3 depicts the insertion of segment of size 2.

4 Priority Rules

In this section a hyper-heuristic is described (HH). The procedure is inspired on the Scatter Search (SS) Meta heuristic [15]. A key difference between HH and the original SS procedure is the following: 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 and its work overload value. That is done using a constructive procedure of rules combination (PCCR).

4.1 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 $R=\{r_1,r_2,r_3,\ldots,r_R\}$ 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.

4.2 Hyper-heuristic

Similarly to SS [15], our proposal (HH) 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 RefSet.
if ( RefSet state is not improved )
    3. Diversify RefSet.
end if
end while
```

Fig. 4. HH pseudo-code

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.

$$cr_r = (r, r, r, ..., r) \rightarrow RSs = \{cr_r : r \in R\} \quad (1)$$

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 process iteration taking into account the new chains with the best values of work overload obtained by the last combination.

$$1 \leq RSd_{it} \leq R \quad (2)$$

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. In Appendix A, are shown the 20 priority rules used in this work. 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 updated *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,* let *cp* and *cq*) are combined, a new one (*son,* let *cr*) 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.

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

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 αR chains in the *RefSet* are regenerated by the αR best chains in the *parents* set.

In the regeneration process, duplication of chains in the *RefSet* 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*. If diversification is needed, 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 obtain (bad) chains containing rules with inferior frequency in the *Fr*, expecting to guide the search in to an unexplored chains space.

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

5 Computational Experiment

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. Instances are designed with different parameter values. To measuring the quality of the solutions obtained by the proposed procedures we use the same global index used by the battery designers.

Originally in [6], 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:

$$rel.wo : \left(\sum_{h=1}^{100} wo_h - \sum_{h=1}^{100} wo^*_h\right) \Big/ \sum_{h=1}^{100} wo^*_h \cdot 100\% \qquad (3)$$

We use two indexes for measuring the aggregated relative deviation. The original index (3) is represented by our index *rel.wo*2. Trying to get more information on the quality of the solutions obtained with the proposed procedures, in our index *rel.wo*1 the value wo^*_h is a lower bound *lbw* (4). Computations are performed in a Pentium 4 CPU 2.4 GHz, 512 MB RAM under a system Microsoft windows XP professional 2002.

$$lbw = \sum_{k=1}^{K} \left[\sum_{i=1}^{I} n_i \cdot p_{ik} - \left(c \cdot (T-1) + L_k\right)\right]^+ \qquad (4)$$

In LS computational experiment, four constructive procedures (CP) from [10] were taken to build a seed sequence for the search: Ud, UdC, UdR and YRx. Table 1 shows the *rel.wo*1 and *rel.wo*2 values obtained after LS with the different neighbourhoods. In the search by three elements sweeping, the computational effort is much bigger than the effort required in 2 elements swap due to the size of the neighbourhood. In Table 1 can be seen an improvement on the indexes when 3S(b) is applied on the local optimal obtained with 2S.

Table 1
Global results after LS.

		*rel.wo*1 (%)				*rel.wo*2 (%)			
		Ud	UdC	UdR	YRx	Ud	UdC	UdR	YRx
LS	Initial	45.64	41.82	40.97	69.25	20.12	16.97	16.37	40.16
Swaps	2S	24.32	24.30	24.53	24.54	2.53	2.51	2.70	2.71
	3S(a)	35.16	33.03	34.84	55.39	11.47	9.72	11.21	28.16
	3S(b)	32.99	31.86	30.55	44.15	9.68	8.75	7.67	18.89
	2-3S(b)	23.96	23.88	24.09	24.06	2.23	2.17	2.34	2.32
Insertions	2 Ins	23.88	23.64	23.80	24.38	2.17	1.97	2.11	2.58
	3 Ins	23.51	23.35	23.34	24.22	1.86	1.74	1.72	2.45
	4 Ins	23.34	23.20	23.19	24.25	1.73	1.61	1.60	2.48
	5 Ins	23.15	23.00	23.00	24.02	1.57	1.45	1.44	2.28
	6 Ins	23.01	22.95	22.99	24.03	1.45	1.40	1.44	2.29
	7 Ins	22.98	22.87	22.77	23.91	1.43	1.34	1.26	2.20
	8 Ins	22.82	22.71	22.63	23.34	1.29	1.21	1.14	1.73
	9 Ins	22.72	22.61	22.60	23.35	1.21	1.12	1.12	1.73
	10 Ins	22.56	22.65	22.44	23.29	1.08	1.16	0.98	1.69

When segment insertion is used, the neighbourhood size is smaller than any of the insertion neighbourhoods (two elements or three elements). Given a segment *or*, its corresponding neighbourhood size is *T-or*. Segment insertion requires less computational effort than swapping. Since the neighbourhoods with swaps can be huge, search time has been limited. Trying to fins a local optimum, different search time limits has been establisher, according to the neighbourhood treated. The limit for LS with swaps was 3600 seconds, except for 2-3S(b), which had 3600 for 2S search, and 1800 seconds for 3S(b) search. In segment insertion search, time limit was 1800 seconds. Other advantage from segment insertion over swaps is the quality of the solutions. In general, the initial seed procedure UdR helps to obtain better results. The best combinations of segment size and initial seed is *or*=10 and UdR. Nevertheless, reader must remember that indexes in table 1 are global indexes which do not differentiate the instances by parameters.

Work overload solutions have been compared with the *lbw* value. Table 2 shows the optimums reached. The best average values are obtained with the 2-3S(b), 3Ins, 4Ins, 5Ins and 6Ins. This time, the difference between the initial procedures is small. Until a quarter of the instances, the *lbw* value is reached with the combination UdR-5Ins.

Table 2
Optimums proved with *lbw*.

Optimums		Initial Procedure				
		Ud	UdC	UdR	YRx	Average
LS	Initial	2	2	2	1	1.75
Swaps	2S	20	19	20	20	19.75
	3S(a)	17	17	13	13	15.00
	3S(b)	16	17	18	17	17.00
	2-3S(b)	22	23	21	24	22.50
Insertions	2 Ins	20	21	20	21	20.50
	3 Ins	23	23	22	22	22.75
	4 Ins	22	22	23	22	22.25
	5 Ins	21	21	25	23	22.50
	6 Ins	23	22	22	23	22.50
	7 Ins	21	22	21	22	21.50
	8 Ins	20	22	20	22	21.00
	9 Ins	23	19	18	21	20.25
	10 In	20	20	20	20	20.00

The number of proved optimums using segment insertion neighbourhood increases when the segment size increases to, until the 5Ins. With this segment size are reached until 25 proved optimums with the initial seed obtained from UdR. For small instances, more optimums are reached when *or* is around *T*/2.

A second computational experiment has been done to test the performance of the hyper-heuristic proposed procedure. The analysis considers the combination of three kinds of regenerations procedures described in subsection 4.2 (Reg1, Reg2, and Reg3), and the maximum number of diversifications permitted (D3-D6). Table 3 shows three different global relative deviation indexes: *rel.wo*1, *rel.wo*2 and *rel.wo*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. Indexes are grouped according to the parameters *I* (5, 10 and 20), *K* (5, 10 and 20) and *L* (110, 150 and variable).

The performance of the three regeneration methods is similar. The more diversifications are applied, the better results are obtained; nevertheless, this improvement is small. Work overload solutions obtained with HH are almost as good as the solutions obtained considering CPLEX. In average, 156 seconds are needed by the HH procedure to finishing the search. HH gets better results than CPLEX in 52% of the instances, in the 11% gets the same result. HH is better than CPLEX in small instances, in medium size instances with different length stations, and in medium size instances with wide windows.

HH gets relatively better *rel.wo*1 values than those obtained with the CP, for instances with $I = 5$, $I = 10$, $K = 5$ and $K = 10$. In instances with $I = 20$, $K = 20$ the seed sequences of the CP have better indexes than those obtained with HH. The regeneration Reg1 gets the smaller index *rel.wo*1 in instances with parameters $I = 5$, $I = 20$, $K = 5$ and $K = 10$. Reg2 works best with *rel.wo*1 in instances which has different station lengths. Reg 3 has the best relative deviation for instances with parameters $I = 10$, $K = 20$, and with instances which have all stations with the same length (groups L=110 and L=150). Basically, in small and medium-small ($I = 10$ and $K = 10$) instances, HH obtain better *rel.wo*1 values than the CP. In 44% of the instances, HH gets better work overload values than the seed sequences (CP). In this context, the instances less favored by HH are those with many products and stations ($I = 20$ and $K = 20$), or instances with different station lengths. In instances with wide windows or with basic product, HH and CP obtain similar results.

Table 3
Relative deviation values by parameters.

Group		*rel.wo*1 (%)			*rel.wo*2 (%)			*rel.wo*3 (%)		
		Reg1	Reg2	Reg3	Reg1	Reg2	Reg3	Reg1	Reg2	Reg3
I = 5	D3	2.31	2.35	2.65	0.05	0.10	0.38	0.87	0.92	1.21
	D4	2.31	2.35	2.65	0.05	0.10	0.38	0.87	0.92	1.21
	D5	2.31	2.35	2.65	0.05	0.10	0.38	0.87	0.92	1.21
	D6	2.31	2.31	2.65	0.05	0.05	0.38	0.87	0.87	1.21
	Avg.	2.310	2.340	2.650	0.050	0.875	0.380	0.870	0.907	1.210
I = 10	D3	32.94	32.90	32.62	0.70	0.67	0.45	2.20	2.17	1.95
	D4	32.86	32.84	32.56	0.64	0.62	0.41	2.14	2.12	1.91
	D5	32.82	32.78	32.52	0.60	0.58	0.38	2.11	2.08	1.88
	D6	32.77	32.74	32.47	0.57	0.55	0.34	2.07	2.05	1.84
	Avg.	32.847	32.815	32.542	0.675	0.605	0.395	2.13	2.105	1.895
I = 20	D3	84.34	88.44	84.32	0.76	0.81	0.75	0.82	0.87	0.81
	D4	84.23	84.41	84.08	0.70	0.80	0.62	0.76	0.85	0.68
	D5	84.00	84.17	83.86	0.57	0.66	0.50	0.63	0.72	0.56
	D6	83.64	83.85	83.86	0.37	0.49	0.50	0.43	0.55	0.56
	Avg.	84.052	85.217	84.030	0.600	0.690	0.592	0.660	0.747	0.652
K = 5	D3	2.31	2.35	2.65	0.05	0.10	0.38	0.87	0.92	1.21
	D4	2.31	2.35	2.65	0.05	0.10	0.38	0.87	0.92	1.21
	D5	2.31	2.35	2.65	0.05	0.10	0.38	0.87	0.92	1.21
	D6	2.31	2.31	2.65	0.05	0.05	0.38	0.87	0.87	1.21
	Avg.	2.310	2.340	2.650	0.050	0.087	0.380	0.870	0.907	1.210
K = 10	D3	26.00	25.90	25.74	0.71	0.63	0.50	2.25	2.16	2.03
	D4	25.93	25.84	25.68	0.65	0.68	0.45	2.19	2.11	1.98
	D5	25.84	25.73	25.61	0.58	0.49	0.39	2.11	2.02	1.93
	D6	24.76	25.65	25.55	0.51	0.42	0.35	2.05	1.95	1.88
	Avg.	25.632	25.780	25.645	0.612	0.555	0.422	2.150	2.060	1.955
K = 20	D3	89.59	89.83	89.32	0.72	0.85	0.58	1.14	1.27	1.00
	D4	89.44	89.78	89.14	0.65	0.83	0.49	1.06	1.24	0.90
	D5	89.39	89.72	89.04	0.62	0.80	0.43	1.03	1.21	0.84
	D6	89.23	89.61	89.04	0.53	0.73	0.43	0.95	1.15	0.84
	Avg.	89.412	89.735	89.135	0.630	0.802	0.482	1.045	1.217	0.895
L = 110	D3	81.03	81.11	80.63	0.94	0.98	0.71	2.01	2.05	1.78
	D4	80.87	81.02	80.48	0.85	0.93	0.63	1.91	2.00	1.70
	D5	80.72	80.85	80.32	0.76	0.84	0.54	1.83	1.90	1.60
	D6	80.54	80.67	80.24	0.66	0.74	0.49	1.73	1.80	1.56
	Avg.	80.790	80.912	80.417	0.8025	0.8725	0.5925	1.870	1.937	1.660
L = 150	D3	0.61	0.61	0.55	0.08	0.08	0.02	0.55	0.55	0.49
	D4	0.59	0.61	0.55	0.06	0.08	0.02	0.53	0.55	0.49
	D5	0.59	0.61	0.55	0.06	0.08	0.02	0.53	0.55	0.49
	D6	0.59	0.60	0.55	0.06	0.07	0.02	0.53	0.54	0.49
	Avg.	0.595	0.607	0.550	0.065	0.077	0.020	0.535	0.547	0.490
[88-132]	D3	10.69	10.47	10.50	0.59	0.40	0.43	2.67	2.48	2.51
	D4	10.68	10.42	10.43	0.59	0.35	0.36	2.67	2.43	2.44
	D5	10.67	10.38	10.43	0.58	0.32	0.36	2.66	2.39	2.44
	D6	10.60	10.38	10.43	0.52	0.32	0.36	2.60	2.39	2.44
	Avg.	10.660	10.412	10.447	0.570	0.347	0.377	2.650	2.422	2.457

LS outperform HH, with the disadvantage of the time required to explore the neighborhoods. In average, HH requires 1/8 of the time needed by LS, and gets the half of the optimums obtained by LS. In general, the bester results are produced by the LS. HH coincide with the best value founded by LS in 14% of the instances. The more of such instances are small, with wide windows and with basic model. In HH, Reg1 obtain the best results. HH is deviated 17.33% from the best result founded with the procedures.

6 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.

In this paper we experiment with both: local search and priority rules. Four seed solutions for LS are gotten by constructive procedures. The LS improves the global relative deviation of the seed solutions to almost the half. Local search with segment insertion outperforms the elements swap neighbourhood, requiring less computational effort. LS finds the best solutions when is combined with the UdR seed constructive procedure.

The hyper-heuristic proposed procedure tries to take advantage of both, the priority rules (problem knowledge) used normally in greedy procedures, and the generations of new good solutions inside the search space. The intensification phase of HH is based on the combination of a set of good rules chains in such a way that those rules appearing with more frequency are repeated in the new generated chains. The frequency of the appearance of each rule in the *RefSet* is also used in the diversification phase of the procedure. Three strategies are used in the intensification of the *RefSet*. Results of each strategy are very similar. The results of HH are almost as good as those that can be obtained with the seed constructive procedures used in LS. The LS gets much better results than HH, but require eight times more effort than HH. Other combination and diversification strategies must be explored to improve the results. A natural extension of this paper is the integration of effective local search procedures in the HH which promises interesting results.

Acknowledgments

We acknowledge Nissan Operations Spain in Barcelona as well as the UPC Nissan Chair for partially funding this research work. This research paper has also been partially funded by DPI2004-03475 grant from the Spanish government. Second author acknowledges the grant from the CONACYT, Mexico.

7 References

[1] J. F. Bard, E. Dar-El, and A. Shtub, An analytic framework for sequencing mixed model assembly lines. International Journal on Production Research, 30:35-48, 1992.
[2] Y. Monden. Toyota production system. Institute of Industrial Engineers Press, Norcross GA, 1983.
[3] C. A. Yano, and R. Rachamadugu. Sequencing to minimize work overload in assembly lines with product options. Management Science, 37:572-586, 1991.
[4] A. Bolat, and C. Yano. Scheduling algorithms to minimize utility work at a single station on paced assembly line. Production Planning and Control, 3:393-405, 1992.
[5] L. Tsai. Mixed-model sequencing to minimize utility work and the risk of conveyor stoppage. Management Science, 41:485-495, 1995.
[6] A. Scholl, R. Klein, and W. Domschke. Pattern based vocabulary building for effectively sequencing mixed-model assembly lines. Journal of Heuristics, 4:359-381, 1998.
[7] W. Zeramdini, H. Aigbedo, and Y. Monden. Bicriteria sequencing for just-in-time mixed-model assembly lines. International Journal of Production Research, 38:3451-3470, 2000.
[8] S. Kotani, T. Ito, and K. Ohno. Sequencing problem for a mixed-model assembly line in the Toyota production system. International Journal of Production Research, 42:4955-4974, 2004.
[9] A. Bolat, and C. Yano. A surrogate objective for utility work in paced assembly line. Production Planning and Control, 3:406-412, 1992.
[10] J. Bautista, and J. Cano. Minimizing work overload in mixed-model assembly lines. International Conference on Industrial Engineering and Systems Management, IESM 2005.
[11] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross and S. Schulenburg. Hyper-heuristics: an emerging direction in modern search technology. F. Glover and G. Kochenberger (eds), Handbook of Metaheuristics, Kluwer, 2003.
[12] E. Burke and S. Petrovic, Recent research directions in automated timetabling. European Journal of Operations Research, 140(2): 266-280,2002.
[13] P. Ross, S.Schulenburg, J. Marin-Blazquez, E. Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2002, Morgan Kauffmann, 942-948, 2002.
[14] D. Wolpert and W. MacReady. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1):67-82, 1997**.**
[15] M. Laguna, and R. Martí. Scatter search, methodology and implementations in C. Kluwer Academic Publishers, USA, 2003.

Appendix A

Let:

p_{ik}	processing time for product *i* in station *k,*
n_i	original demand for product *i,*
T	total production or sequence length, $T = \sum n_i$,
d_i	pending demand for product *i,*
r_{ik}	allocation dynamic index for product *i* in station *k*, $r_{ik} = d_i \cdot \Delta_{ik}$,
Δ_{ik}	worker movement in station *k* due to product *i* $\Delta_{ik} = \mid p_{ik} - c \mid$,
L_k	length of station *k*,
u_i	ideal production for product *i*, $u_i = n_i / T$,
s_k	initial worker position in station *k*,
w_t	accumulated work overload until position *t* in the sequence, $w_t = \sum_{\tau=0}^{t-1} w_{\tau k}$,
$\overline{p}$	average processing time $\overline{p} = \sum p_{ik} /(K \cdot I)$,
k^b	bottleneck station,
w_i	total work overload in the current position due to the assignment of product *i*, $w_i = \sum_k w_{ti}$,
o_i	total idle time in the current position due to the assignment of product *i*, $o_i = \sum_k o_{ti}$.

The following are the 20 priority rules used in the HH procedure:

R1. Product with the biggest processing time (PT), $i^* \in \arg\max\{p_{ik}\}$

R2. Product with the smallest PT, $i^* \in \arg\min\{p_{ik}\}$

R3. Product with the PT closest to average PT, $i^* \in \arg\min \{|p_{ik} - \overline{p}\,|\}$

R4. Product with the PT closest to *c*, $i^* \in \arg\min\{ \Delta_{ik} \}$

R5. Product with the biggest pending demand, $i^* \in \arg\max \{ d_i \}$

R6. Product with the smallest pending demand, $i^* \in \arg\min \{ d_i \}$

R7. Product with the biggest dynamic index, $i^* \in \arg\max \{ r_i \}$

R8. Product with the smallest dynamic index, $i^* \in \arg\min \{ r_i \}$

R9. Product that produces the biggest total workers movement, $i^* \in \arg\max \{\sum_{k=1}^{K} \Delta_{ik} \}$

R10. Product that produces the smallest total workers movement, $i^* \in \arg\min \{\sum_{k=1}^{K} \Delta_{ik} \}$

R11. In the bottleneck station, the product with the biggest PT, $i^* \in \arg\max \{ p_{ik^b} \}$

R12. In the bottleneck station, the product with the smallest PT, $i^* \in \arg\min \{ p_{ik^b} \}$

R13. Product closest to ideal production, $i^* \in \arg\min \{| (n_i - d_i) - t \cdot u_i |\}$

R14. Product with the biggest total dynamic index (in all stations), $i^* \in \arg\max \{\sum_{k=1}^{K} r_{ik} \}$

R15. Product with the smallest total dynamic index (in all stations), $i^* \in \arg\min \{\sum_{k=1}^{K} r_{ik} \}$

R16. Product that produces the biggest overload in the current period, $i^* \in \arg\max \{ w_i \}$

R17. Product that produces the biggest idle time in the current period, $i^* \in \arg\max \{ o_i \}$

R18. Product that produces the smallest overload in the current period, $i^* \in \arg\min \{ w_i \}$

R19. Product that produces the smallest idle time in the current period, $i^* \in \arg\min \{ o_i \}$

R20. Under the concept of regularity, the product with the overload closest to ideal overload, assuming the lower bound if the instance is *lbw* > 0,

$$i^* \in \arg\min \left\{ \left| \left(lbw / T \right) \cdot t - w_t - w_i \right| \right\}$$

where w_t is the accumulated overload.