A Scatter Search Based Hyper-Heuristic for Sequencing a Mixed-Model Assembly Line

Jaime Cano-Belmán
Graduate Program in Systems Engineering
Universidad Autónoma de Nuevo León, Mexico
jaime@yalma.fime.uanl.mx

Roger Z. Ríos-Mercado
Graduate Program in Systems Engineering
Universidad Autónoma de Nuevo León, Mexico
roger@mail.uanl.mx

Joaquín Bautista
Nissan Chair
Universitat Politècnica de Catalunya, Spain
director@nissanchair.com

27 February 2009

Abstract

In this paper a mixed-model assembly-line sequencing problem with the goal of minimizing work overload is addressed. We consider time windows in work stations of the assembly line (closed stations). Different versions of a product to be assembled in the line (e.g. automotive industry), which require different processing time according to the work required in each work station are considered. In a paced assembly line, products are feeded in the line at a predetermined constant rate (cycle time). Then, if many products with processing time greater than cycle time are feeded consecutively, work overload can be produced when the worker have insufficient time to finish its job. A scatter search based hyper-heuristic is proposed for this NP-hard problem. The procedure implies the use of priority rules. Computational results that show the efficiency of the hyper-heuristic are presented.

Keywords: Just-in-Time scheduling; assembly line; priority rules; work overload; scatter search; hyper-heuristic

1 Introduction

Assembly lines are commonly used in many environments such as the automotive industry, for instance. Two important decisions for managing mixed-model assembly lines are to spread out work in stations (balancing) and to determine the sequence for introducing cars into the assembly line (sequencing). When the mid-term decision of balancing has been taken, a sequencing decision must be considered. Depending on the manufacturing environment it may be desirable to minimize or maximize some parameters or characteristics of the line [2]. One of the two main criteria [23, 10] for sequencing mixed models on assembly lines aims at leveling the load on stations. The second one searches for the leveling part usage. The problem addressed in this paper is closely related to the first objective. In particular we are interested in the reduction of the work exceeding the work station capacity which may be called differently according to the system characteristics and the enterprize management philosophy: *work overload* [37, 38], *remaining work* [6], *utility work* [4, 33, 17], or *conveyor stoppage* [35, 13].

The problem addressed in this paper considers the objective of minimizing work overload. Since processing time for a product in a station can be greater 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 jobs. 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 and downstream limits. 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 work based on this criterion was carried out by Yano and Rachamadugu [38] and Bolat and Yano [8]. Exact methods have been proposed (Xiaobo and Ohno [35]) to minimize the total conveyor stoppage time for instances with 16 total products. Bolat [7] minimizes the total cost of production for 100-job and 10-station problems.

Nevertheless, due to the complexity of the problem, heuristic approaches are more common than exact methods to minimize work overload and other sequencing objectives. Some of these heuristics consider not only the minimization of work overload [37, 38, 8, 9, 4], but multiple objectives [39, 21, 15]. Also, meta-heuristics ([35] minimize work overload, or [30] minimize conveyor stoppage time), multi-objective meta-heuristics ([17] minimize utility work, keep a constant rate of part usage and minimizing setup cost with genetic algorithms; [19] keep a constant rate of part usage, smooth production load and minimize setup times with simulated annealing; [31] minimize utility work cost, product rate variation cost and setup cost with a memetic algorithm), hybrid multi objective heuristics ([25] minimize utility work, production rate variation and setup cost), and beam search ([16] part-usage variation and load leveling) have been developed.

Lately researchers have rescued a heuristic concept originated during the 60's, *evolving heuristics*. Nowadays, the concept is better know as *hyper-heuristics*. This kind of heuristics is an effort to get more general framework optimization systems [34] which can lead to obtain good enough - soon enough - cheap enough solutions to a problem, instead of designing a height-specific, knowledge-based meta-heuristic [11].

Some problems in which the hyper-heuristics performance has been tested are the timetabling and rostering problems [12] or the bin-packing problem [26]. As it normally occurs in scheduling problems, many rules or characteristics of the problem may influence in the procedure for solving it. The problem addressed in this paper can also consider different surrogated rules such as idle time, bottleneck station, product demand, or worker displacement. Thus, considering a set of problem-related priority rules, and looking for a whole search algorithm, a hyper-heuristic is proposed to construct the mixed-model assembly line sequence (MMALS). The objective is to minimize the work overload. Such hyper-heuristic is developed inside of a well known meta-heuristic: Scatter Search (SS). SS is a population-based meta-heuristic [22] which has been applied successfully to many combinatorial problems such as clustering [28], p-median [14], and scheduling [36], for instance.

This paper is organized as follows. In Section 2 the sequencing problem studied in this paper is described and a formulation is presented. Section 3 presents some of the more relevant and recent works related to our problem and approach. The scatter search-based hyper-heuristic and its elements, including the use or priority rules related to the problem are fully described in Section 4. The use of priority rules related to the problem in the hyper-heuristic environment is also described. An empirical evaluation of the proposed solution approach is given in section Section 5. Conclusions are presented in Section 6.

2 Problem Description

In pure continuous production systems, the issue of programming normally focuses on the allocation of operations to workplaces, the establishment of the route, and the balancing of the line. But, if several products with small variations are assembled in a line it is also necessary to establish the sequence of these products. This happens for example in a final automobile assembly line.

In pull production management systems, specifically, within the framework of the philosophy of just-in-time production, the sequence of operations becomes the key aspect of operations programming. This kind of problem can be characterized as a permutation problem with objectives represented by some efficiency measure. A representative productive system is an assembly line. Two main activities in the efficient management of an assembly lines are the balancing and the sequencing. Thus, once the medium to long term decision of balancing has been implemented, and the production mix is known, the next step is to establish a sequence for introducing units in the line according to some criteria.

In this paper we consider that there are products requiring processing times greater than the cycle time. This is due to the fact that the mixed-product line balancing is done with the condition that the operation time of each process is not greater than the cycle time. Cycle time is the elapsed time between the entry of two consecutive products to the line (or any station). If products with processing time greater than cycle time are introduced in the line successively, a delay in the completion of one of these products can be produced in some station of the line. There will be excessive work load in the line. This excessive work in the stations can be found in the literature as *overload* [37, 38]), *remaining work* [6], or *utility work* [4, 33, 17]. When the management philosophy of the company is comited to complete the work within the stations boundaries without receiving extra help, the term *line stoppage* is used (Xiaobo and Ohno [35]; Celano et al. [13]), as the workers have an obligation to halt the line in order to complete work on products inside the station boundaries. Throughout this paper the term *work overload* is used.

2.1 Operative Characteristics and Assumptions

The following are the characteristics of the production system and assumptions of the problem addressed in this paper. General surveys or different versions of the problem can be found, for example, in Bard, Dar-El, and Shtub [2] or Boysen, Fliedner, and Scholl [10]. Normally, the sequencing problem appears in product-oriented systems. Such systems converge into a final assembly line.

The needed operations to assembly a product are done sequentially. Different products are produced (product $i = 1, \ldots, I$) and all of them follow the same route in the final assembly line. There exist a great variety of product components, sources and services in the stations. Under a just-in-time environment, the variety of product components creates the need of determining a product sequence. The assembly line has been previously balanced. The lay-out of the line, the number of stations (K) and their length (L_k) are fixed ($k = 1, ..., K$).

For the period considered, the product demand n_i for each product i is known. The total of products is $T = \sum_{i=1}^{I} n_i$. The expected demands have been considered in the balancing phase. The assembly line balancing is determined on the basis of the expected mix so that, on average, the expected work load in each station can be completed within the station boundaries. All the limitations of the line and of the strange processes have been previously considered during the production program elaboration.

The system production rate is established by both the conveyor speed and the launching frequency, assumed known. Both are fixed, then a paced assembly line is considered. Products on the assembly line are equidistant from each other. The time between two consecutive launchings is called cycle time (c), and it has been determined during the balancing phase. There are processing times p_{ij} for each product i in each station k. Processing times are deterministic and include the

setup times.

The stations considered in this problem are closed. In this kind of stations, the jobs on products can only be done within the station boundaries due to some tool or equipment restriction. The station length L_k is the maximum available time to do a job on a product in station k. There are no buffers between stations. A product is maintained in a station k at least c time units ($L_k \geq c$). It is assumed that processing time p_{ik} does not exceed the available time L_k in station k.

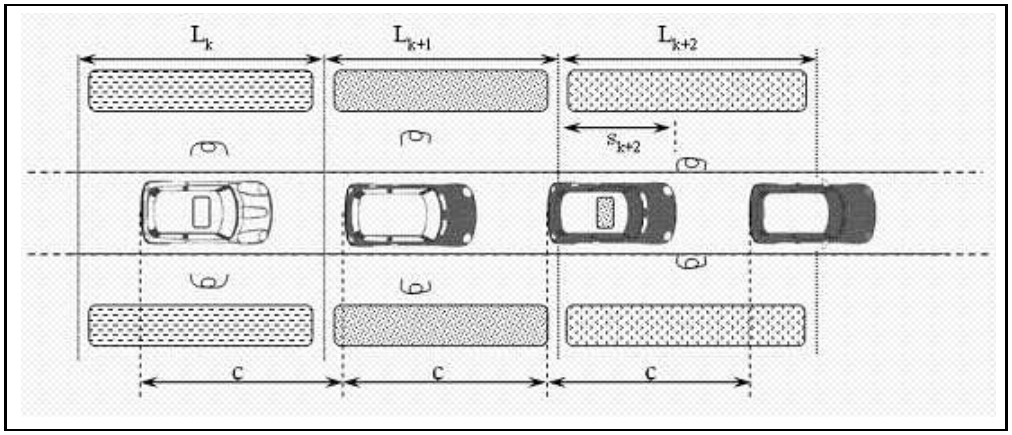

Figure 1: Assembly line diagram.

Figure 1 depicts three consecutive closed stations with length L_k. Two workers do the job on the product simultaneously. If the available time is not enough to finish the product before it leaves the station, utility workers assist in order to finish the job inside the station boundaries. It is assumed that there are enough utility workers. Workers do the job as soon as possible. The time a worker needs to reach a new product is negligible.

2.2 Work Overload

Yano and Rachamadugu [38] proposed a general formulation for measuring work overload. Work overload is measured in time units and the time unit is the cycle time $c = 1$. Authors define s_t as the starting time of the job on product in position t of the sequence ($t = 1, \ldots, T$), and f_t the finishing time of the job in position t. If only one stations is considered $s_t = \max(t-1, f_{t-1}))$ and $f_t = \min(s_t + p_t, t - 1 + L)$, where p_t is the processing time of the product in position t of the sequence. Without loss of generality $s_1 = 0$. Thus, given a sequence π of T elements and considering one station, the work overload w_t obtained in position t of the sequence is $w_t = [s_t + p_t - (t-1+L)]^+$ where $[x]^+ = \max(0, x)$. The total work overload is $w(\pi) = \sum_{t=1}^{T} w_t$.

2.3 Formulation

Under the above mentioned system characteristics and assumptions, formulations of the problem can be found in Scholl, Klein, and Domschke [30]. The model considers all the distances can be measured in time units, which can be assumed due to the paced conveyor. The model given in [30] takes into account the conditional on the final position of the worker on stations, which must be the origin of their station in position $T+1$ of the sequence. We do not consider such condition on our problem.

A feasible solution is a sequence containing the demanded n_i units of product i. An optimal solution minimizes the work overload produced in all the stations. A sequence can be described by the binary variable x_{it} which takes the value 1 if a product i is assigned in position t of the sequence, 0 otherwise. Two auxiliary variables are included; s_{kt} to measure the position of the worker in period t in station k, and w_{ik} to indicate the work overload produced in station k after processing the unit in position t. It is assumed all the worker start the job on the first product at the upstream station limit ($s_{k1} = 0$: $k = 1, \ldots, K$). Variables s and w can be computed as shown in (1) and (2), respectively.

$$s_{kt} = \max\{\min\{s_{k,t-1} + \textstyle\sum_i p_{ik} \cdot x_{i,t-1}, L_k\} - c, 0\} \quad k = 1, \ldots, K; t = 2, ..., T \tag{1}$$

$$w_{kt} = \max\{s_{kt} + \textstyle\sum_i p_{ik} \cdot x_{it} - L_k, 0\} \quad k = 1, \ldots, K; t = 1, ..., T \tag{2}$$

Following Scholl, Klein, and Domschke [30] and Scholl [29], a simplification of the processing time required to carry out the job on product in position t in station k can be represented by $\rho_{kt} = \sum_{i=1}^{I} p_{ik} \cdot x_{it}$. Thus, the problem can be formulated as a mixed-integer programming model as follows:

$$\text{Minimize } Z(x, w, s) = \sum_{k=1}^{K} \sum_{t=1}^{T} w_{kt} \tag{3}$$

$$\text{Subject to } \sum_{i=1}^{I} x_{it} = 1 \qquad t = 1, \ldots, T \tag{4}$$

$$\sum_{t=1}^{T} x_{it} = n_i \qquad i = 1, \ldots, I \tag{5}$$

$$s_{kt} + \rho_{kt} - w_{kt} - c \leq s_{k,t+1} \quad k = 1, \ldots, K; \quad t = 1, \ldots, T-1 \tag{6}$$

$$s_{kt} + \rho_{kt} - w_{kt} \leq L_k \quad k = 1, \ldots, K; \quad t = 1, \ldots, T \tag{7}$$

$$s_{kt}, w_{kt} \geq 0 \quad k = 1, \ldots, K; \quad t = 1, \ldots, T \tag{8}$$

$$s_{k1} = 0 \qquad k = 1, \ldots, K \tag{9}$$

$$x_{it} \in \{0,1\} \quad k = 1, \ldots, K; \quad t = 1, \ldots, T \tag{10}$$

The objective function (3) minimizes the total work overload. Constraints (4) ensure that only one item is assigned in each position of the sequence. The satisfaction of the product demand is guaranteed by (5). Constraints (6) ensures that in station k, a job can not be started until the previous job has been finished. The condition that jobs are completed within the station limit is met by (7). The nature of the decision variables is given in (8) and (10). The position of the worker on the upstream station limit, in period 1 of the sequence is represented by (9). In this model the starting times are related only to each station, without considering the finishing times in the previous stations. The variable s_{kt} can also be considered as the lost time with respect of the ideal instant in which a job could had been started in the next position. The problem is NP-hard [38, 30, 21].

Let Π be the collection of permutations of $T = \sum_{i=1}^{I} n_i$ items from which n_i are of kind i. The problem, from the combinatorial optimization perspective, consists of finding a permutation $\pi = (\pi_1, \ldots, \pi_T) \in \Pi$ satisfying the specified workers displacement constraints (12), that minimizes the associated work overload (11).

$$\underset{\pi \in \Pi}{\text{Minimize}}\, w(\pi) = \sum_{k=1}^{K} \sum_{t=1}^{T} w_{k,\pi_t} \tag{11}$$

$$\text{Subject to } \pi \in F = \{\pi : s_{k,\pi_t} + p_{k,\pi_t} - w_{k,\pi_t} - c \leq s_{k,\pi_{t+1}}, s_{k,\pi_t} + p_{k,\pi_t} - w_{k,\pi_t} \leq L_k\} \tag{12}$$

The variables s_{k,π_t} and $w_{k,\pi t}$ represent the worker position in station k when the job on product in position π_t of the permutation is started and the work overload produced in station k after the job on product in position π_t has been finished, respectively. Such variables can be computed recursively with (13) and (14), assuming $s_{k,\pi_1} = 0$: for $k = 1, \ldots, K$.

$$s_{k,\pi_t}(\pi) = \max\{\min\{s_{k,\pi_{t-1}} + p_{k,\pi_{t-1}}, L_k\} - c, 0\} \quad k = 1, \ldots, K; t = 2, \ldots, T \tag{13}$$

$$w_{k,\pi_t}(\pi) = \max\{s_{k,\pi_t} + p_{k,\pi_t} - L_k, 0\} \quad k = 1, \ldots, K; t = 1, \ldots, T \tag{14}$$

2.4 Example

In order to illustrate the work overload problem, a single station 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) respectively. 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.

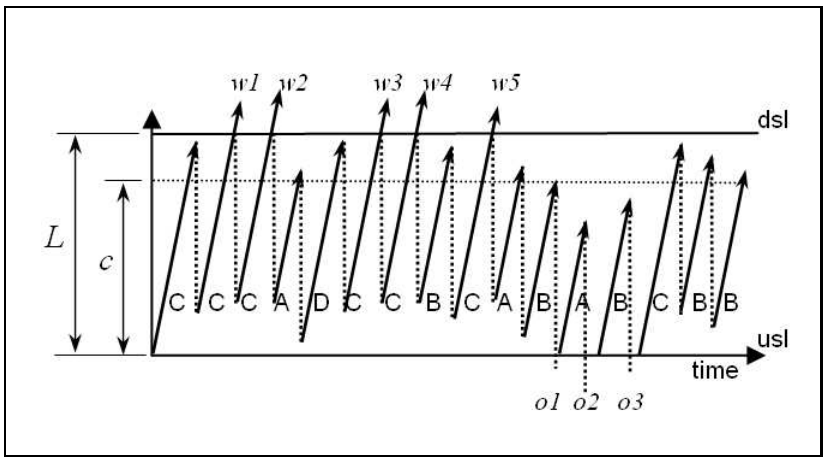

Figure 2: Worker movement diagram.

Without loss of generality, the initial worker position is assumed to be on the upstream limit of the station. Figure 2 represents the movement of the worker during his/her 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 downstream 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 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 (usl) and would finish it 0.19+1.19 (starting time + processing time) units away from the usl. 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/her 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 for finishing 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 usl. This time, the job is completed. Products in positions 6, 7, and 9, also produce work overload ($w3 = 0.16$, $w4 = 0.19$, $w5 = 0.13$). When the worker finishes the job on product of kind B in position 11, she/he is 0.98 units away from the usl and goes back to reach the unit in position 12 (product A), but the worker must wait 0.02 ($o1$) time units until the product get in the station. Recall a new product get in a station each cycle time $c = 1$. The job on product A is started when it is on the usl and finished 0.0 + 0.82 units away from usl. Thus, in position 12 of the sequence, the worker is idle 1 - 0.82 time units ($o2$) while waiting product B in position 13 to get in the station. Similarly, finishing product B in position 13 produces 0.06 units of idle time ($o3$). The total work overload produced by the sequence is 0.85 cycle time units. The total idle time is 0.26 cycle time units.

3 Related work

Several performance measure have been studied to manage efficiently mixed model assembly lines. The earliest work dates back to Thomopoulos [32] who studies the sequencing problem by looking for even flow of work along the line. Then, Monden [23] remark two main objectives to deal with model sequencing on assembly lines: level the load, and keep a constant usage of parts. Other classifications are given, for example, in [37] and [10]. A mathematical framework is given by Bard, Dar-El, and Shtub [2] for six variations of the sequencing problem on mixed model assembly lines.

Work load related sequencing problems have been solved optimally. Xiaobo and Ohno [35] proposed two algorithms to minimize total conveyor stoppage time: branch-and-bound and simulated annealing methods. The first can find optimal solutions for instances with less than 16 products. Bolat [7] minimizes the total cost of production satisfying human resource capacity limits at a stations. Optimal solutions for 100-jobs and 10-stations problems, with a proposed branch-and-bound procedure, which employs some dominance criteria, are found.

An exact algorithm is proposed by Bolat [6] to minimize die total amount of remaining work. The algorithm can solve 20-job 8-station instances, but requires a prohibitive quantity of computational effort and memory allocation.

A formulation of the mixed-model sequencing problem to minimize work overload is given by Scholl, Klein, and Domschke [30]. In general, the system characteristics described by the authors are the same as the ones described for the problem studied here. Authors consider an additional condition: the state of the initial and final state (after producing all units) of the system must be the same ($S_{kt} = S_{k,T+1} = 0$). A truncated branch-and-bound procedure, with a deep-first-search strategy is used. The branching sub-problems order is controlled by different priority rules. Lower bounds are obtained by relaxing the problem to a transportation problem. Authors argue that due to the relatively weak bounds obtained is not appropriate to apply more elaborate bounds arguments based on the TSP-relaxation considered. An informed tabu search with pattern based vocabulary building strategies to minimize the work overload is also proposed (the word *informed* is used to allude tabu search utilizes the information coded within a vocabulary building strategy).

Since the computational effort and memory requirements to solve real-size problems with exact methods is extremely large, heuristic and meta-heuristic algorithms have been proposed to solve work overload sequencing problems.

To minimize work overload, a prospective heuristic is proposed in Yano and Bolat [37]. Yano and Rachamadugu [38] show a mathematical formulation for the sequencing problem, with the aim of minimizing the work overload. Two products are considered: basic and optional, with processing time lower and larger than cycle time, respectively. The proposed algorithm for K-station models is based on an single-station exact algorithm. The greedy K-station algorithm uses a a lower bound of the objective function to sequentially determine which job should be placed in the next position.

Bolat and Yano [8] proposed various procedures. One of them is based on the use of space rules. Two more greedy procedures are described. Using dynamic programming, the best conditions of each heuristic are evaluated. A simple hyper-heuristic is proposed in Bautista and Cano [3] to minimize the work overload in a single station with optional products. An extension of existing procedures is proposed for multi-product and multi-station problems.

Bolat [5] approaches the problem using minimum jobs sets. Optimal and heuristic algorithms are proposed. Bolat [6] gives an stochastic algorithm that combines simulated annealing and specific knowledge methods to minimize utility work.

3.1 Related Criteria

Okamura y Yamashina [24] proposed a heuristic algorithms to minimize the risk of stopping the conveyor, that is, minimizing the maximum distance a worker would need to proceed downstream from his/her work station until all the work has been completed. In Bolat and Yano [8] the aim is to minimize the total units with option above the k allowed units in sequence stretches of size l. To minimize the total conveyor stoppage time, Xiaobo and Ohno [35] propose branch-and-bound and a simulated annealing procedures to minimize conveyor stoppage time.

3.2 Multi-Objective Approaches

Approaches considering the work overload criteria together with other mixed-model assembly-lines related criteria can be found in the literature. Considering open and closed stations, Sharker and Pan [27] minimize total utility time cost and idle time cost. Tsai (1995) [33] considers the minimization of both: risk of conveyor stoppage (minimize maximum displacement), and utility work. He derives an $O(\log T)$ optimal algorithm for both objectives for a single station and two kinds of jobs (basic and optional).

Aigbedo and Monden [1] propose a parametric procedure that takes into account four objectives simultaneously: smoothing of parts utilization, smoothing of product load, product rate variation smoothing, and sub-assembly load variation smoothing. Erel, Gocgunz, and Sabuncuoğlu [16] developed six beam search algorithms to level both part usage and workload simultaneously. The procedure is an adaptation of the branch-and-bound method in which a limited number of solutions paths are explored in parallel. Authors explain that the method has the ability to use the optimizing properties of the branch-and-bound method with limited number beams. Nevertheless, if the number of beams gets large, computational requirements increase significantly.

Hyun, Kim, and Kim [17] give a mathematical formulation with three objectives: minimizing total utility work, keeping a constant rate of part usage, and minimizing total setup cost. A genetic algorithm that offers near-Pareto or Pareto optimal solutions is proposed. A genetic evaluation and selection mechanism, called Pareto stratum-niche cubicle, is also given.

Using simulated annealing in Kim and Cho [19], near optimal solutions are obtained with three objectives: to keep a constant rate of usage of all parts used by the line, to maintain a smooth production load, and to minimize the sequence-dependent setup times.

Three objective are studied in Rahimi-Vahed and Mirzaei [25]: total utility work, total production rate, and total setup cost. An hybrid multi-objective algorithms based on shuffled frog-leaping algorithms and bacteria optimization is proposed.

3.3 Two-Phase Procedures

Bolat [4] analyzes the *trim* line sequencing and a paint station. Two kind of jobs are considered. The procedure proceeds in two phases. First, a look-ahead procedure is used to minimize the utility work on the *trim* line. In a second phase, trim and paint stations are treated together incorporating the setup cost of changing a job color in the paint shop.

The two main goals of keeping a constant rate of usage of parts and smoothing the workload at work stations to avoid line stoppages are studied in Zeramdini, Aigbedo, and Monden [39]. In a first of two phases, the rate of part usage is solved with an extension (look-ahead version) of the goal chasing method given by Monden [23]. In the second phase, spacing constraints are used to deal with the workload. The same objectives are considered in Korkmazel and Meral [20]. Existing approaches are used to minimize the sum of deviations. The best approaches are extended to the second objective. Kotani, Ito, and Ohno (2004) [21] give a formulation for minimizing the total line stoppage time with auxiliary workers. A second goal of keeping a constant rate of part usage is included in the model as constraints. A two-phase approximation algorithm is proposed.

Korkmazel and Meral [20] study the level of loads in stations and keep a constant part usage. In a first phase, well-known approaches for constant part usage are analyzed. In the second phase the best approaches are joined with the load objective and the problem is formulated as an assignment problem which obtain optimal solutions for small-sized problems. Taking into account the problem characteristics and assumptions described above and the related works (Bolat and Yano [8]; Scholl, Klein, and Domschke [30]; Yano and Rachamadugu [38]) the following hyper-heuristics is proposed to deal with the reduction of the work overload on assembly lines.

4 Proposed Heuristic

In this section our proposed hyper-heuristic (HH) is described. The goal of a hyper-heuristic is to raise the level of generality at which an optimization system can operate. In the literature, a HH is described as a heuristic that chooses other heuristics to solve a problem [12]. In the proposed procedure, the Scatter Search meta-heuristic is used as the high level meta-heuristic. In the lower level of the procedure, priority rules are used. These priority rules are used to build solutions using a constructive procedure which we call $PCCR$.

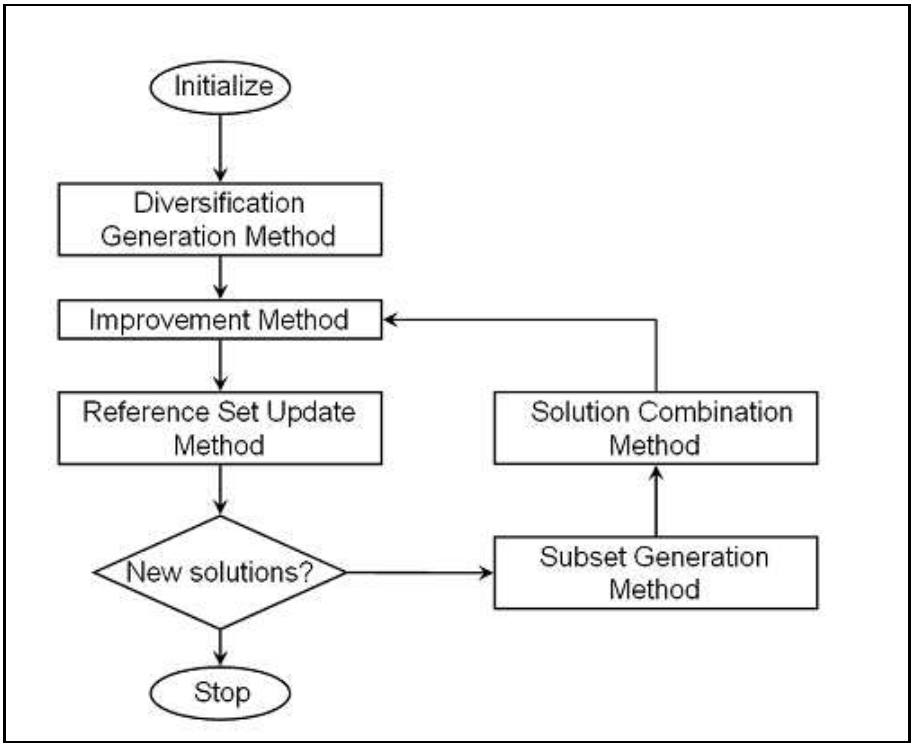

Figure 3: Basic Scatter Search scheme.

Scatter Search (SS) is a population-based metaheuristic procedure consisting mainly of five methods: a Diversification Generation Method (DGM), an Improvement Method (IM), a Reference Set Update Method (RSU), a Combination Method (CM), and a Subset Generation Method (SGM). The details of some of these method depend on the specific characteristics of the problem. Normally, a scatter search procedure (see Figure 3) starts with the DGM. This method creates diverse solutions in the space search. This solutions are stored in the set *pool*. The next step is to apply the IM to all the solutions in *pool*. Then the RSU method selects good solutions from the set *pool* and store them in a Reference Set ($RefSet$). Generally, the $RefSet$ is divided in a subset of solutions with best objective values and a subset with diverse solutions. The following step consists on selecting subsets of solutions in the $RefSet$ (SGM) to combine them with the SCM. The new solutions obtained with the combination method can also be improved with the IM. The $RefSet$ is updated again with the best solutions (quality and diverse). This methods interact iteratively until no new (better) solution is found. In the proposed HH we aim to keep the advantages of the combination strategies used in SS to produce new elements from elite elements in the $RefSet$.

4.1 PCCR

PCCR is a greedy constructive procedure based on the combination of priority rules. Priority rules are commonly used in solving scheduling problems in combination with problem knowledge. In this

procedure PCCR (C, n_i)
Input: n_i = demand for product i; a rule chain $C = r^1, r^2, \ldots, r^T$.
Output: an equivalent solution $S = s^1, s^2, \ldots, s^T$.
0 $S = \emptyset$, $d_i = n_i$;
1 **for** ($t = 1$ to T) **do**
2 $s^t \leftarrow i^* \in r^t$;
3 Update d_{i^*};
4 **end for**
5 **return**(S);
end PCCR.

Figure 4: PCCR procedure

paper $|R|$=20 problem-related priority rules are used. The work overload value for a priority rule chain (C) is obtained by applying the $PCCR$ in Figure 4.

Figure 4 shows the PCCR procedure. The inputs of the PCCR are a priority rule chain $C = \{r^1, r^2, \ldots, r^T\}$, where r^t is the rule in position t of the chain, and the original product demands n_i. The rule in position t of C determines from a set of candidates, the product i^* that best satisfies the rule r (see also Figure 5). Product i is a candidate if $n_i > 0$. This product i^* is assigned in position t of the product sequence, and its pending demand is updated. Having a product sequence, its work overload value can be computed.

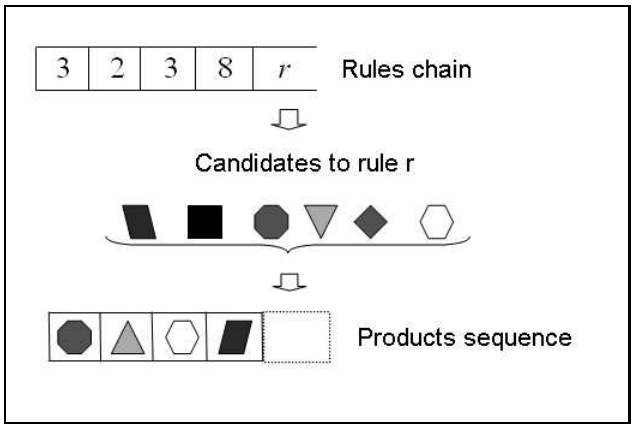

Figure 5: Constructive procedure by rules combination.

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 lexicographic order, until the tie disappears. Different criteria are considered in the priority rules used in the procedure. In the Appendix, the 20 priority rules used in this work are stated. 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 largest and smallest 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 selects according the regularization of idle time.

4.2 Hyper-Heuristic

Similarly to SS in Laguna and Martí [22], our proposed HH is a population-based algorithm that creates new elements combining the existing ones. A key element in the SS methodology is the $RefSet$, which stores good solutions and diverse solutions along the search process. Those solutions are combined in order to create new solutions. Basically, a SS implementation requires a generator of diverse solutions, a method to improve solutions, a method to update the elements in the $RefSet$, a combination method, and a method to select the solutions to be combined.

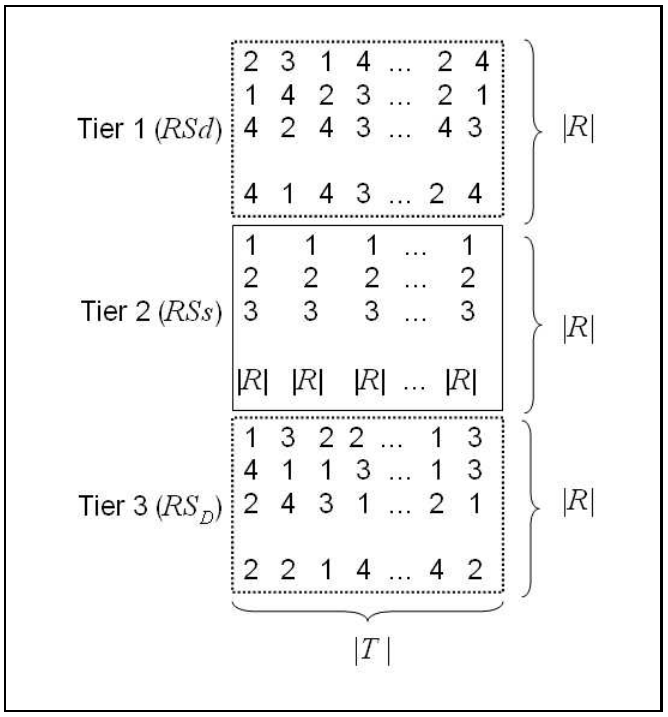

Figure 6: Reference set with 3 tiers.

The proposed HH also operates on a reference set. But, instead of a reference set of solutions, we use a reference set of rule chains C. Combining those chains in the reference set, new chains are created. Normally, a $RefSet$ is a collection of both b_1 high quality solutions and b_2 diverse

solutions that are used to generate new ones using a combination method [22]. The size of the reference set is $b = b_1 + b_2$. See Figure 6.

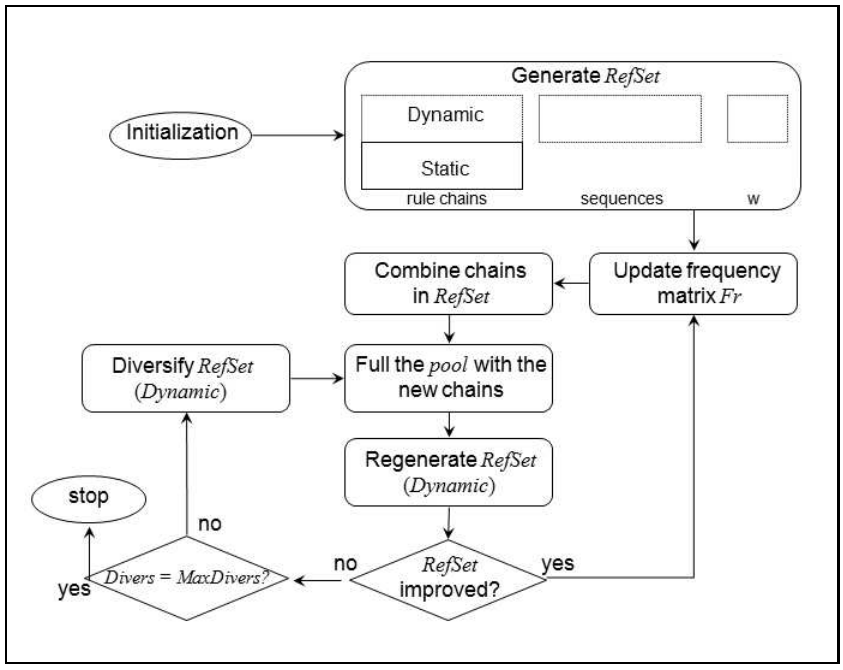

Figure 7: HH1.

In HH, we use two different chain reference sets, which we henceforth call RS1 and RS2. When any differentiation of a reference set is needed, it is just called $RefSet$. RS1 uses two tiers: a dynamic tier (RSd) and a static tier (RSs). The size of RSd and RSs is $b_1 = |R|$ and $b_3 = |R|$, respectively. RSd stores chains that produced good quality product sequences. RSd is initialized with rules $r \in R$ randomly. The tier RSs is filled at the beginning of the procedure with chains that only contain one priority rule. This tier is called static, because it is not modified during the search process. Thus, in RSs the chain 1 contains only the rule 1, the chain 2 contains only the rule 2, and so on. The reason to use a static tier is to ensure that all the priority rules are considered in the combination phase along the entire search.

RS2 uses a diverse tier (RS_D) as well. RS_D contains $b_2 = |R|$ diverse solutions. This tier is also initialized randomly. In HH, a normal diversification generator method is not used. As it has been mentioned, the initial reference set is filled randomly (RSd and RS_D) and systematically (RSs). Since RS2 include diverse elements, an explicit diversification phase is not needed. Both schemes are shown in Figure 7 and Figure 8, respectively. In Figure 7, the $RefSef$ contains a good quality elements tier, and a static tier, then the process requires an explicit Diversification phase. In Figure 8, the $RefSet$ has a third tier for diverse elements, and the Diversification phase is not required.

Figure 9 shows a pseudo-code of the proposed procedure when RS1 is used. The procedure starts

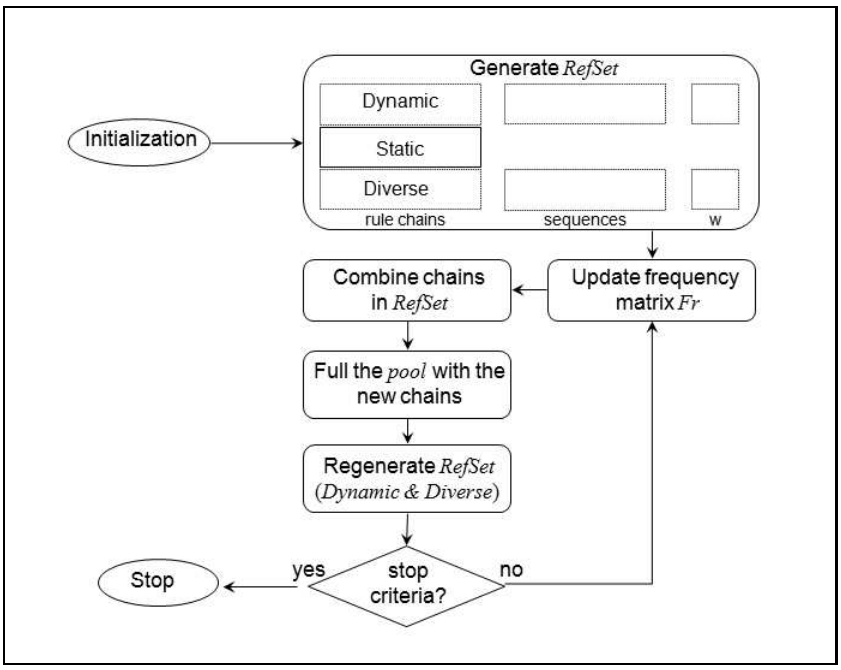

Figure 8: HH2.

computing information needed along the search. In step 1, RSd is randomly initialized and RSs is systematically initialized. During step 2, the equivalent product sequences of RSd are obtained and its work overload value is computed. A frequency matrix Fr is updated in step 3. Fr stores the number of times a rule r has been in position t of a chain. In step 6, the chains in the $RefSet$ are combined with the combination method. The Fr matrix is a key element in the combination method. All the combined chains are introduced in the set $Pool$. In step 7, the corresponding sequences and work overload values of the chains in $Pool$ are computed. Step 8 is optional. If high quality solutions are preferred, the improvement method is applied on the sequences in $Pool$. Depending on the CPU time available, local search can be applied at different grades of depth. In step 9, the $RefSet$ is regenerated with our reference set update method. The worst $b_1/2$ chains are substituted with the best $b_1/2$ chains in $Pool$. Since steps 6-9 are intensification steps, in step 10, the $RefSet$ state is verified. If $RefSet$ has not been improved with new chains (step 10), it is diversified in step 11. Step 13 updates the Fr matrix and *stop criteria*. Finally, when the stopping criteria are reached (maximum time or maximum number of diversifications without improvement), the procedure leaves the while loop and apply local search on sequences in the last $RefSet$ obtained (step 15). Finally, the procedure returns the best solution found S_{best}.

The general pseudo-code for HH with RS2, is shown in Figure 10. The differences with the previous procedure are the following: the $RefSet$ contains a diverse tier as well, which is initialized randomly. Because of this, in step 10, the diversification process is not required because it is implicit in the combination method. Thus, in step 10, the stopping criteria are just evaluated and updated.

procedure HH1 (R, $Stop\_criteria$, $Improve\_kind$)
Input: R = set of priority rules considered; $Stop\_criteria$ = criteria to stop search process; $Improve\_kind$ = depth of improvement applied on *pool* sequences.
Output: best solution found S_{best}.

0 Setup priority rules info; $stop = False$; $Fr = \emptyset$; $RefSet = \emptyset$; $Pool = \emptyset$;
1 Initialize $RefSet$;
2 Get sequences and values for chains in $RefSet$;
3 S_{best} = best sequence in $RefSet$;
4 Update frequency matrix Fr;
5 **while** (!stop) **do**
6 Combine chains in $RefSet$;
7 Get $Pool$ chains, sequences and values;
8 Improve sequences in $Pool$ acording to $Improve\_kind$;
9 Regenerate $RefSet$;
10 Update S_{best} = best sequence in $RefSet$, if necessary;
11 **if** ($RefSet$ state is not improved) **do**
12 Diversify $RefSet$;
13 **end if**
14 Update Fr;
15 **if** ($Stop\_criteria$) **do** $stop = True$;
16 **end while**
17 Find local optimum for sequences in RSd;
18 Update S_{best} = best sequence in $RefSet$;
19 **return**(S_{best});
end HH1.

Figure 9: Hyper-heuristic scheme 1.

procedure HH2 (R, $Stop\_criteria$, $Improve\_kind$)
Input: R = set of priority rules considered; $Stop\_criteria$ = criteria to stop search process; $Improve\_kind$ = depth of improvement applied on *pool* sequences.
Output: best solution found S_{best}.

0 Setup priority rules info; $Fr = \emptyset$; $RefSet = \emptyset$; $Pool = \emptyset$;
1 Initialize $RefSet$;
2 Get sequences and values for chains in $RefSet$;
3 S_{best} = best sequence in $RefSet$;
4 Update frequency matrix Fr;
5 **while** (!stop) **do**
6 Combine chains in $RefSet$;
7 Get $Pool$ chains and sequences;
8 Improve sequences in $Pool$ according to $Improve\_kind$;
9 Regenerate $RefSet$;
10 Update S_{best} = best sequence in $RefSet$, if necessary;
11 **if** ($RefSet$ state is not improved $\|$ $Stop\_criteria$) **do**
12 stop = TRUE;
13 **end if**
14 Update Fr;
15 **end while**
16 Find local optimum for sequences in RSd;
17 Update S_{best} = best sequence in $RefSet$;
18 **return**(S_{best});
end HH2.

Figure 10: Hyper-heuristic scheme 2.

In the following sections the components of the hyper-heuristic are described in detail.

4.3 Combination Method

This method combine chains with the purpose of creating new chains. For each pair of chains in the $RefSet$ a new combined chain is obtained. Thus, given two parent chains $C_p = [r_p^1, r_p^2, \ldots, r_q^T]$ and $C_q = [r_q^1, r_q^2, \ldots, r_q^T]$ a new chain $C_k = [r_k^1, r_k^2, \ldots, r_k^T]$ is obtained according to (15). In other words, the rule in the position t of the combined chain is determined according to the frequency that the rule r has in the position t in the $Fr(r,t)$ matrix.

$$C_k^t = \begin{cases} C_p^t & \text{if } C_p^t = C_q^t \\ C_p^t & \text{if } Fr(C_p^t, t) \geq Fr(C_q^t, t) \\ C_q^t & \text{otherwise} \end{cases} \tag{15}$$

$Fr(r,t)$ is a frequency matrix containing the number of times that a rule r has appeared in the position t of the chains in the RSd. Since RSs do not change, it is not necessary consider it for computing Fr. For example, let $C_p^t = 4$, $C_q^t = 2$, $Fr(4,t) = 5$ and $Fr(2,t) = 8$, then in position t of the new chain will be the rule 2, because $Fr(2,t) > Fr(4,t)$.

The idea behind the combination method is that the $RefSet$ contain an elite set of chains, and that the extrapolation of the rules with more frequency in some positions of the chains of this set, can guide the search to obtain better results.

The combined chains are stored in the $Pool$. When the $Pool$ is fulled with the combined chains, the PCCR is used in order to get the work overload value for each chain. The equivalent sequence can then be improved with the improvement method described next.

4.4 Improvement Method

Usually, an improvement method is needed if good quality outcomes are desirable. Nevertheless, a ßprocedure can be implemented without it. In HH both cases are considered. The improvement method is not applied on the priority rules chains, but on their equivalent product sequences.

Given a chain C, its equivalent product sequence is obtained with the PCCR procedure. Such product sequence can be represented by the permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_T)$ of a total of T objects from which n_i objects are of kind i, thus, all solutions are guaranteed to be feasible. Given a solution S representable by π, the neighborhood $N(S)$ consist of all solutions reachable from S by moving a segment of seg consecutive elements of S from its current position t in the sequence to a different position $t_{next} \neq t$. Such a move is denoted by $move\_seg(t, t_{next})$. See Figure 11.

The segment size seg in HH is restricted to $1 \leq seg \leq 15$. In instances with $T \leq 15$, the maximum value for seg is 8. Given T and a segment of size seg, all the possible moves are $|N(S)| = (T - seg)^2$. Every time a neighborhood is going to be explored, a segment size seg

is selected randomly, having more possibility to chose those segment sizes that have previously produced improvement in the local search.

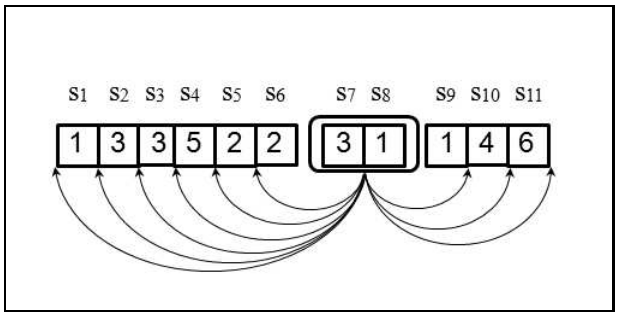

Figure 11: Segment insertion.

In HH, improvement is applied on two specific points of the process: in the *Pool* sequences, and in the final $RefSet$ sequences. In *Pool*, there are a lot of sequences that can be improved, which requires a relatively large amount of time if a local optimum is wanted. Thinking in saving time in *Pool* sequences improvement, only one $N(S)$ is evaluated. A work overload lower bound *lbw* can be easily obtained with (16) and can be used to determine an improvement percentage with respect to the original solution value before the improvement (i.e. 10%, 20%). In the other hand we have b_1 sequences in the final $RefSet$. Since b_1 is much smaller than $|Pool|$, it could be worth find a local optimum in such sequences.

$$lbw = \sum_{k=1}^{K} \left[\sum_{i=1}^{I} n_i p_{ik} - (c(T-1) + L_k) \right]^{+} \tag{16}$$

4.5 Reference Set Update Method

In each iteration of the process, the $RefSet$ is updated. Once all the combined chains and their equivalent sequence have been obtained (and optionally, improved), the chains with better values are considered for updating the RSd. In other words, the RSd is regenerated. The regeneration process for RS1 is shown in Figure 12. Basically, the worst $b_1/2$ chains in the RSd are substituted by the best $b_1/2$ chains in *Pool*.

When the RS2 is used, the RS_D must be updated as well. The procedure is shown in Figure 13. Once the RSd has been updated in steps 1-8, chains in diverse tier RS_D are deleted in step 9. Iteratively, for each chain in *Pool* the distance $d(c, RSd, RS_D)$ is computed. Such distance measure is the well-known Hamming distance (Jain, Murty, and Flynn [18]). The chain $c \in Pool$ is compared with all the chains in RSd and with the current chains in RS_D. In step 11, the chain with the maximum distance is selected, and introduced in the RS_D (step 13).

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

procedure Regenerate I (RS, $Pool$)
Input: $Pool$ = current $Pool$; RS = current RS.
Output: Regenerated RS.

0 Introduced= 0; Get $Pool_{best}$; Get RSd_{worst};
1 **while** ($Introduced < b_1/2$) **do**
2 **if** ($Pool_{best} \notin RSd$) **do**;
3 Substitute RSd_{worst} by $Pool_{best}$;
4 $Pool = Pool \setminus Pool_{best}$;
5 Update $Pool_{best}$ and RSd_{worst};
6 $Introduced$++;
7 **end if**
8 **end while**
9 **return**(Regenerated RS);
end Regenerate I.

Figure 12: Regeneration of RS1.

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.

4.6 Diversification Phase

The diversification process is applied only when the reference set RS1 is used. The diversification is needed when the state of the $RefSet$ has not been improved during the last $NoImproveMaxCombs$ iterations.

The pseudo-code of the diversification procedure is shown in Figure 14. In step 1, the diversification process follows the same steps of the combination method. Nevertheless, the key difference is that this time the criteria to select a rule r from two parent chains, is not to take the one with more frequency in Fr but the one with less frequency. Given two parent chains C_p and C_q, one diversified chain C_k is obtained.

$$C_k^t = \begin{cases} C_p^t & \text{if } C_p^t = C_q^t \\ C_p^t & \text{if } Fr(C_p^t, t) < Fr(C_q^t, t) \\ C_q^t & \text{otherwise} \end{cases} \tag{17}$$

The idea is to obtain chains containing rules with inferior frequency in the Fr, expecting to guide the search to an unexplored space.

procedure Regenerate II (RS, $Pool$)

Input: $Pool$ = current $Pool$; RS = current RS.

Output: Regenerated RS.

0 $Introduced = 0$; Get $Pool_{best}$; Get RSd_{worst};
1 **while** ($Introduced < b_1/2$) **do**
2 **if**($Pool_{best} \notin RSd$)**do**;
3 Substitute RSd_{worst} by $Pool_{best}$;
4 $Pool = Pool \setminus Pool_{best}$;
5 Update $Pool_{best}$ and RSd_{worst};
6 $Introduced$++;
7 **end if**
8 **end while**
9 $Introduced = 0$; $RS_D = \emptyset$;
10 **while** ($Introduced < b_2$) **do**
11 Compute $d(c, RSd, RS_D)$ for all $c \in Pool$;
12 $RS_D \leftarrow RS_D \cup c^* = \max\{d(c, RSd, RS_D)\} : c \in Pool$;
13 $Introduced$++;
14 **end while**
15 **return**(Regenerated RS);

end Regenerate II.

Figure 13: Regeneration of RS2.

procedure Diversification (RS, $Pool$)
Input: RS = current $RefSet$; $Pool$ = current $Pool$.
Output: diversified $RefSet$.
0 $Pool = \emptyset$;
1 Combine all chains in $RefSet$ according to (17);
2 Store all chains, their sequences and wo value in $Pool$;
3 $RSd = \emptyset$;
4 $RSd \leftarrow RSd \cup Pool_{best}$;
5 $Introduced = 1$;
6 **while** ($Introduced < b1$) **do**
7 Compute $d(c, RSd)$ for all $c \in Pool$;
8 $RSd \leftarrow RSd \cup c^* = \max\{d(c, RSd)\} : c \in Pool$;
9 $Introduced{+}{+}$;
10 **end while**
11 **return**(diversified $RefSet$);
end Diversification.

Figure 14: Diversification procedure.

Once a $Pool$ with diversified chains has been obtained, the next step of the diversification process iteratively looks for the more diversified chains. That is, the process identifies those chains in $Pool$ that are more 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 (Hamming distance). A coincidence exists if in the same position t, both of the chains have the same rule r.

5 Empirical Work

For evaluating the heuristic performance, we use the set of instances from [30] for mixed-model assembly lines. For each instance $c = 90$ 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. A summary of the data set are given in the Figure 1. There are 9 groups with particular characteristics. The first column indicates an identifier for the group, the second column (NInsts)contains the number of instances in the group. Columns 3 and 4 show the number of models and stations, respectively. Column 5 refers to station lengths. The last column indicates the total of products in the instance and the number of instances with exactly these characteristics. Small instances are in group 1 and 2, considering 5 models and 5 stations. All instances in group 1 have station length of 110 time units, whereas group 2 station length varies in the range [88,132]. Group

Table 1: Data set characteristics

Group	NInsts.	I	K	L_k	T
1	5	5	5	110	10(2), 20(2), 30(1)
2	5	5	5	[88,132]	10(2), 20(2), 30(1)
3	30	10	10	110	10, 80, 110, 140, 170, 200, 250, 300, 350, 400 (3 each)
4	10	20	10	110	140, 200, 250, 300, 350 (2 each)
5	10	10	20	110	140, 200, 250, 300, 350 (2 each)
6	10	20	20	110	140, 200, 250, 300, 350 (2 each)
7	10	10	10	[88,132]	140, 200, 250, 300, 350 (2 each)
8	10	10	10	150	140, 200, 250, 300, 350 (2 each)
9	10	10	10	110	140, 200, 250, 300, 350 (2 each)/basic model

7 also considers different station lengths, but with 10 models and 10 stations. Group 3 is conformed by 30 instances, and the largest instances with 400 products are in this group. Group 4 have twice as many models than stations, whereas group 5 has instances with double quantity of stations than models. Group 8 is the heaviest in the sense that considers 20 models and 20 stations. Group 8 has the largest number of stations, and group 9 considers a basic model with fixed station lengths. A basic model is an austere version of the product, which requires a relatively small job quantity, unless one or more optional components with one or more attributes have to be assembled to it. For more details on the data generation we refer the reader to [30]. Thus, the instance bed consists of 100 instances of small, medium, and large sizes, and different levels of load due to station length, processing times, and number of products and stations.

To measure the quality of the solutions s obtained for instance h, the deviation measure dev is used (18), where s_{best} is the best solution found for instance h using some of the procedures. We use this measure because we only know 25 optimum values of the 100 instances. Even when we can get a lower bound with (16), or get it from CPLEX (e.g. after one hour of search), we do not know certainly the quality of this bound even for small instances. Furthermore, it is known that for some instances (i.e. unloaded instances), the solution value can be $wo_h = 0$. CPLEX obtain the GAP for MIP problems in the same way. Computations have been performed in a PC with processor Pentium 4 CPU 2.4 GHz, 512 MB RAM under a system Microsoft Windows XP professional 2002.

$$dev_h = |s_h - s_{best}|/|s_h + \epsilon| * 100\% \tag{18}$$

The two proposed HH procedures are compared with other three procedures: BC, LS, and CPLEX. BC are four approximation algorithms proposed in [3]. Only the best results obtained with those algorithms are used in the comparison. Using the results of the algorithms proposed

Table 2: Results for HH1 and HH2

				HH1				HH2			
Statistic	BC	LS	CPX	NoI	IRSf	IP1N	IP10%	NoI	IRSf	IP1N	IP10%
Av. $dev(\%)$	20.80	1.02	21.06	18.69	1.93	0.08	1.76	20.57	1.63	0.54	0.31
opt*	4	25	18	16	23	25	25	11	24	24	25
AvIter	-	-	-	50.96	50.96	6.24	6.37	10.8	10.8	2.7	1.3
AvDivs	-	-	-	4.95	4.95	0.38	0.48	-	-	-	-

in [3] as seeds, a local search (LS) procedure is applied. LS is truncated after 1800 seconds of search if no local optimum has been found.

The local search scheme used is the one described in section 4.4. CPLEX (version 9.0) was used with a time limit of 1 hour. The deviation of the best integer solution found is reported. Regarding the hyper-heuristic, in both, HH1 and HH2, improvement is applied in two different points of the process. This is done in order to know where the local search produce better results. Thus, the decision maker can evaluate the time-quality factors and find a good compromise. The improvement is applied on sequences of the $Pool$ and the final RS. Thus, we identify four levels of improvement in HH1 and HH2: NoI, $IRSf$, $IP1N$, and $IP10\%$. In NoI no improvement is applied. $IRSf$ apply local search on final dynamic RS sequences obtained with NoI. In $IP1N$ the improvement is directed on sequences of the $Pool$, exploring only one neighborhood of each sequence (given a segment size seg, there are $(T - seg)^2$ possible movements). $IP10\%$ also works on $Pool$ sequences, where the local search is stopped when value of the sequence before the improvement w_0 is improved in $(w_0 - lbw)/10$ units. Recall lbw is a lower bound obtained with expression (16).

As explained in Section 4.6, some parameters are used to determine when the process must diversify or stop the search in HH1. The maximum number of iterations without improvement in the $RefSet$ before applying a diversification phase is $NoImprovemaxcombs = 3$. Normally in SS when no improvement is obtained in the $RefSet$ state in one iteration, the search is stopped. The maximum number if Diversifications (without improvement) permitted before stopping the search is $MaxDivs = 4$. These values were determined in preliminary testing, in which it was observed that more diversifications did not help improve the results significantly. Then, more diversifications do not necessarily help the search process. In HH2, the maximum number of iterations without improvement is $NoImprovemaxcombs = 4$. A maximum CPU time is established to 3600 seconds for the iterations loop.

Table 2 shows the measures used to compare the procedures: average deviation (Av. $dev(\%)$), number of proved optimums reached (opt*), average of iterations for HH1 and HH2, and average of diversifications for HH1. The smallest average deviation is obtained with HH1, using the improvement level $IP1N$. It is followed by HH2 with $IP10\%$. Except by $IP10\%$ in HH1, all the procedures

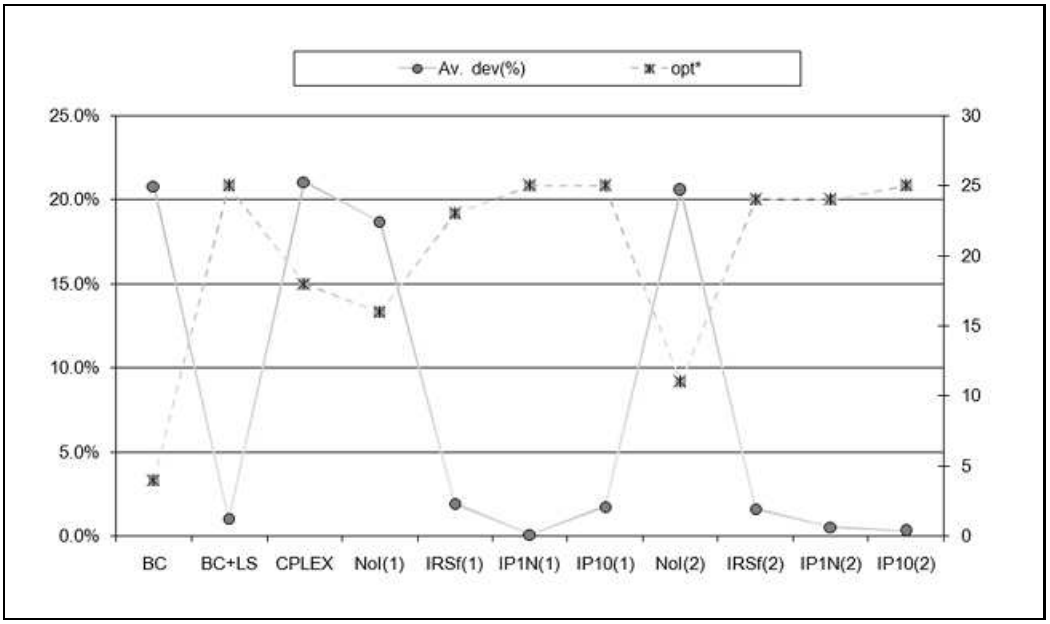

Figure 15: Average deviation and number of proved optimums by procedure.

which apply improvement in *pool* sequences obtain average deviations under 1%. Nevertheless, the CPU time limit is reached in HH1 for 79% of the instances with $PM1N$ and 83% with $PM10\%$. Similarly for HH2, $PM1N$ and $PM10\%$ exceed the CPU limit in 80% and 90% of the instances, respectively. HH1 and HH2 without improvement (NoI) require in average 269 and 511 seconds, respectively. HH1 and HH2 with improvement in the final $RefSet$ ($IRSf$) sequences require 505 and 555 seconds average, respectively, which suggests that que quality of the solutions produced with HH2 are better than those of HH1, since local optimum is reached in lower time. This can also be seen in the average relative deviation values in the table. The isolated local search procedure (LS) also offers good results with 1.02 of average deviation and reaching 25 of the 27 proven optimums known. HH1-$IP1N$, HH1-$IP10\%$, and HH2-$IP10\%$ also achieve 25 proved optimums. Notice, in HH1, that the average deviation value for IP1N is smaller than the one in IP10%. This can be due to the CPU time required by IP10% in which it improves (while the CPU time limit is not exceeded) just a few solutions, instead of, just like happens with IP1N, improving for a few time, many solutions. Figure 15 depicts the average relative deviation and the number of confirmed optimums reached by each procedure. Average deviation values are represented by circles and $opt*$ is represented by squares.

A good compromise between quality solutions and CPU time is offered for $IRSf$, mainly in HH2 scheme. In this sense, HH2-$IRSf$ reaches 24 confirmed optimums in 9.25 minutes average. During one hour of search, CPLEX finds 18 optimums, two more than HH1-NoI in 4.49 minutes average of search, and 7 more than HH2-NoI in 8.52 minutes average. $AvIter$ and $AvDivs$ show the average of iterations and diversifications used by the procedures, respectively. As can be observed, the

Table 3: Average deviation (%) by groups

				HH1				HH2			
Group	BC	LS	CPLEX	NoI	IRSf	IP1N	IP10%	NoI	IRSf	IP1N	IP10%
1	3.98	0.00	0.00	1.02	0.52	0.00	0.00	1.38	0.52	0.00	0.00
2	1.88	0.00	0.00	0.43	0.31	0.00	0.14	2.25	0.21	0.00	0.00
3	21.01	1.12	20.17	19.74	2.03	0.06	1.17	22.61	1.92	0.62	0.30
4	30.18	2.61	35.59	32.97	3.76	0.30	4.73	35.51	4.10	0.82	0.96
5	16.48	0.99	19.42	18.53	1.35	0.05	0.88	20.57	0.83	0.19	0.27
6	17.72	2.02	27.22	23.14	1.64	0.07	2.82	24.15	1.33	0.59	0.60
7	6.19	0.58	53.51	7.76	0.48	0.00	0.37	8.57	0.36	0.07	0.03
8	0.43	0.00	0.03	0.48	0.00	0.00	0.00	0.61	0.00	0.05	0.00
9	71.01	0.63	61.92	44.07	5.52	0.17	3.53	46.63	3.47	1.17	0.34

number of average iterations and diversifications required by $LS1N$ and $LSPool10$ to terminate the process is much smaller then those needed by NoI and $IRSf$. This means that it pays off to invest the extra time improving the sequences in $Pool$.

Since each group of instances has different characteristics it is of particular interest a comparison between groups and procedures. The *dev* values by groups are contained in Table 3. The good global performance of the procedure $IP1N$ for HH1, is maintained by groups. HH1-$IP1N$ find the best results for groups 1, 2, 7, and 8. In fact, all the proven optimum values are known for all the instances in group 1, 2, and 8. The three largest instances are contained in group 3 (with 30 instances) which obtain the best average deviation with the same procedure. Group 9 is a special one. According to the *bestobjval* obtained from CPLEX in 1 hour of search, the lower bound is zero for seven of the ten instances of this group. Notice in HH1 by groups, that the average relative deviation in IP1N always is better than IP10%. The local search in the pool sequences for IP10%, spends more effort in improving a few solutions, meanwhile in IP1N, exploring just in the neighborhood, it explores more solutions with less intensity, but reducing the average relative deviation.

Regarding the segment size used in the local search phase, in order to know if its size is related to the sequence size we performed the following analysis. We analyze the data of the procedure $LS1NPool$ because it got better solutions. Initially, all the segment sizes considered in the process have the same probability to be chosen. When a segment size obtains an improvement its probability to be chosen in future improvements, increases. At the end of the process we have a frequency of segment sizes. Instances have different sequence lengths T: 10, 20, 30, 50, 80, 80, 110, 140, 170, 170, 200, 250, 300, 350 and 400. The segment sizes used take a value in the range $[1, 15]$.

Let θ_{seg} the number of times a segment of size seg improved a solution along the procedure, $\Theta =$

$\sum_{seg=1}^{seg=15} \theta_{seg}$ the total of improvements obtained along the search, and $\tilde{\theta}_{seg} = \theta_{seg}/\Theta$ the percentage of improvements obtained with segment size seg. We can now try to compare $\tilde{\theta}_{seg}$ for different instances (with different T values). Let $\bar{\theta}_{seg,T}$ the average of the percentage of improvements obtained with segment size seg in instances with length T. A scatter plot of those $\bar{\theta}_{seg,T}$ values is shown in Figure 16. The higher the peaks in the plot, the more improvements this seg and T produced in the procedure. One can find high tips through all the surface without indicating some clear tendency or combination of seg and T to produce good $\bar{\theta}_{seg,T}$ values.

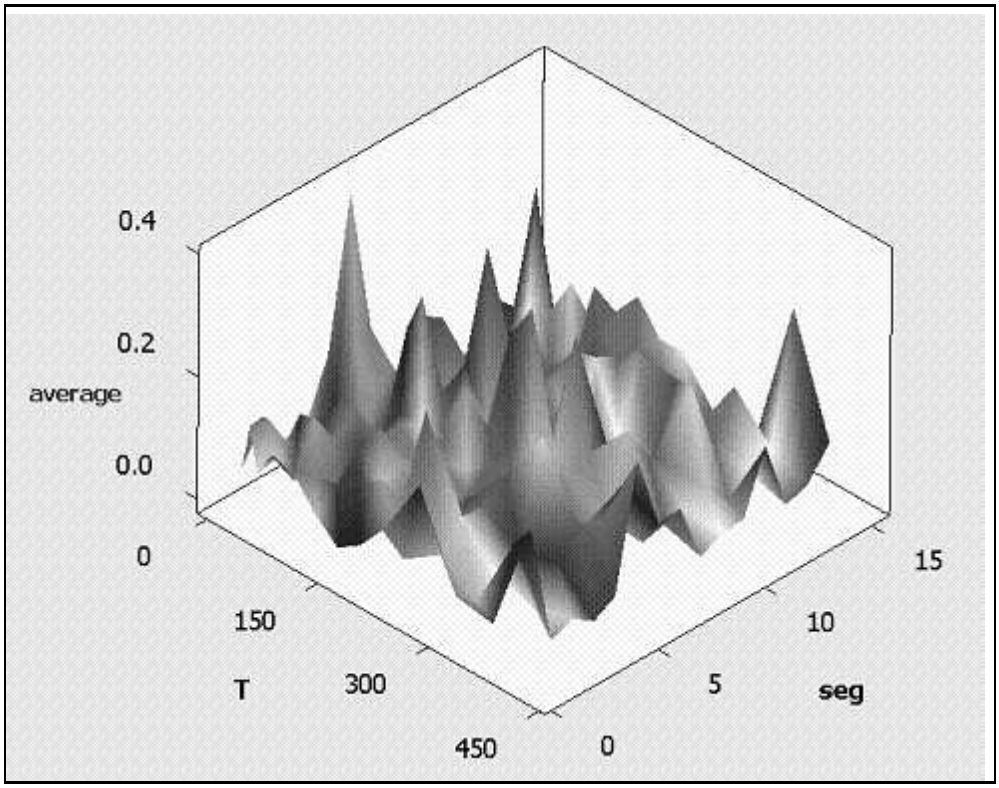

Figure 16: Segment size vs. sequence length.

A multiple regression analysis was done with two independent variables: segment size (seg) and sequence length (T). The response variable is $\bar{\theta}_{seg,T}$. The analysis of variance reports a P regression value = 0.791. The regression analysis is $\bar{\theta}_{seg,T} = 0.0724 - 0.0000T - 0.00071$. We also did the regression analysis only with segment size seg. The analysis reports 11 unusual observations, a P regression value = 0.493 in the analysis of variance, and a regression equation $\bar{\theta}_{seg,T} = 0.0724 - 0.00071seg$. Based on this information we can only mention that the best segment size seg for a given instance depends on the instance characteristics.

Regarding rule frequency at the end of the search, it was clearly observed how some rules resulted more beneficial than others. In Figure 17 the frequency of the rules in each position of the sequence of size $T = 50$ is depicted. This is the smallest instance of group 3. It can be seen many

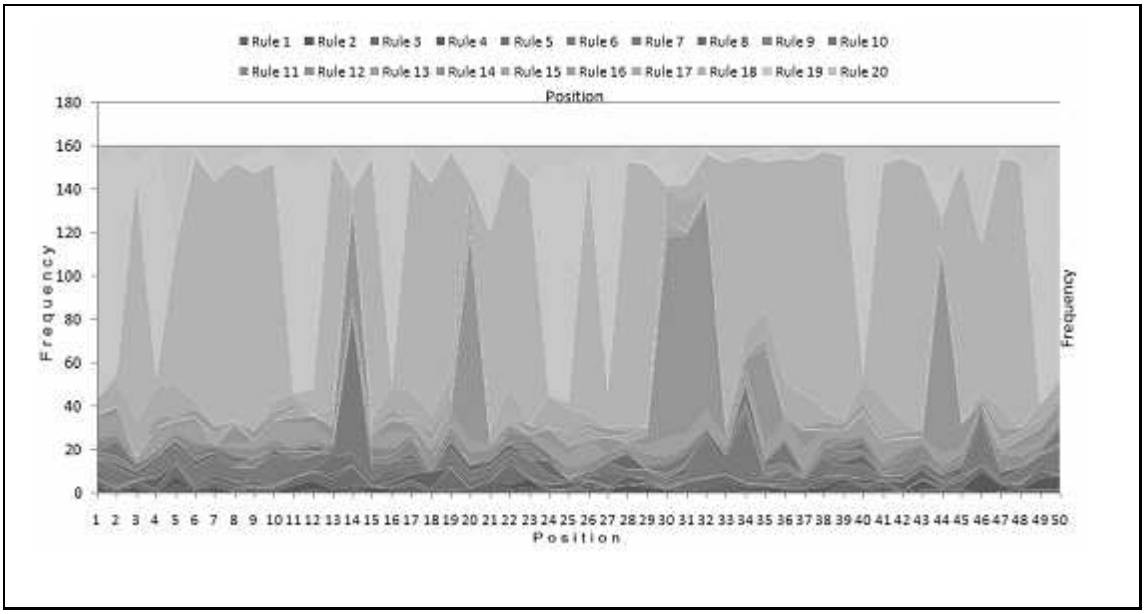

Figure 17: Area graph for rule frequency in each position of the sequence for instance with T=50.

rules have a small frequency value along the sequence, but some of them have clear domain along the sequence. For this instance, four rules appear in more positions in the sequence. The main one is rule 18, which has 44.5% of the frequencies. Rule 18 is the one that selects the product with the lowest overload in the current phase of the sequence. The following rule is the number 19, related to product producing the lowest idle time. Rule 19 has 18.3% of the total frequency in the final Fr. Other rules with significant frequency value are 7, 13, 14, 17, and 20, with 3.5%, 3.1%, 8.5%, 4.6%, and 4.1%, respectively, of the total frequency in final Fr. The dominance of the rules may be better appreciated in Figure 18 which shows the box-plot for each of the 20 rules. Even though there are some dominating rules, there is not a single rule that consistently dominates the other in every position in the sequence. For instance, in Figure 17 one can see how rule 19 dominates in positions 1 and 2, whereas rule 18 dominates in position 3. This combination of rules emphasizes the relation among the main rules that consider overload, idle time, worker displacement, and ideal production.

6 Conclusions

This work addresses a version 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

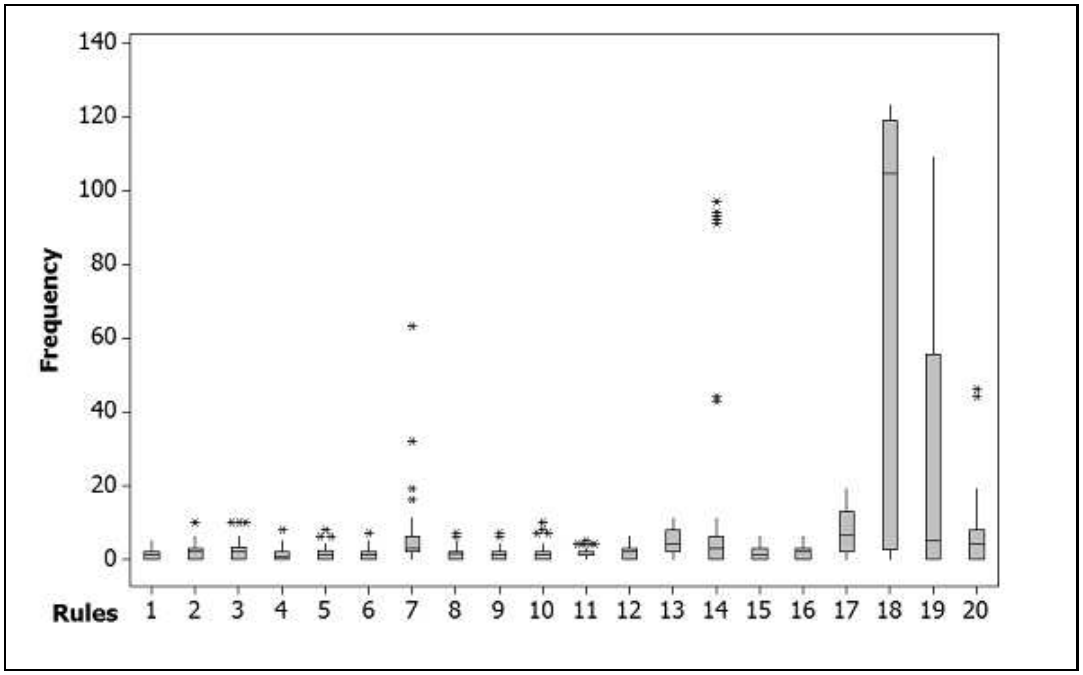

Figure 18: Boxplot for rule frequency for instance with T=50.

worker takes from one product to the next, is negligible. A hyper-heuristic based on a scatter search scheme and the use of priority rules are proposed and compared with procedures from literature. As in scatter search, the hyper-heuristic use a reference set to store elite elements and obtain better ones through the combination of these elements. A key difference is that the reference set contain priority rules chains instead of solutions to the problem. The combination of these chains is based on a rule frequency matrix. Different levels of improvement are applied on sequences of the *pool*, and local search is used to improve sequences on the final reference set. The proposed HH gets the best average relative deviation when improvement is applied in *pool* sequences. A good trade-off between of CPU time and solution quality is observed when local search is used on final reference set sequences only. Final rule frequency matrix suggests the elimination of some rules that do not contribute in the search process significatively. This reduces the $referenceset$ size and the *pool* size, reducing also the CPU time. When compared to existing work, the proposed procedures outperformed the others, particularly with HH1 when improvement is applied for one neighborhood of each sequence in *pool*, reaching all the known proved optimum values.

Acknowledgements: The research of the first author was supported by a Postdoctoral Fellowship from the Mexican National Council for Science and Technology (CONACYT). Also, we thank the UPC Nissan Chair as well as the Spanish Government for partially funding this work by means of PROTHIUS-II project: DPI2007-63026 including EDRF fundings.

References

[1] H. Aigbedo and Y. Monden. A parametric procedure for multicriterion sequence scheduling for just-in-time mixed-model assembly lines. *International Journal of Production Research*, 35(9):2543–2564, 1997.

[2] J. F. Bard, E. Dar-El, and A. Shtub. An analytic framework for sequencing mixed model assembly lines. *International Journal of Production Research*, 30(1):35–48, 1992.

[3] J. Bautista and J. Cano. Minimizing work overload in mixed-model assembly lines. *International Journal of Production Economics*, 112(1):177–191, 2008.

[4] A. Bolat. Sequencing jobs on an automobile assembly line: Objectives and procedures. *International Journal of Production Research*, 32(5):1219–1236, 1994.

[5] A. Bolat. Efficient methods for sequencing minimum job sets on mixed model assembly lines. *Naval Research Logistics*, 44(5):419–437, 1997.

[6] A. Bolat. Stochastic procedures for scheduling minimum job sets on mixed model assembly lines. *Journal of the Operational Research Society*, 48(5):490–501, 1997.

[7] A. Bolat. A mathematical model for selecting mixed models with due dates. *International Journal of Production Research*, 41(5):897–918, 2003.

[8] A. Bolat and C. A. Yano. Scheduling algorithms to minimize utility work at a single station on a paced assembly line. *Production Planning & Control*, 3(4):393–405, 1992.

[9] A. Bolat and C. A. Yano. A surrogate objective for utility work in paced assembly lines. *Production Planning & Control*, 3(4):406–412, 1992.

[10] N. Boysen, M. Fliedner, and A. Scholl. Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373, 2009.

[11] E. K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, chapter 16, pages 457–474. Kluwer, Boston, 2003.

[12] E. K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2):266–280, 2002.

[13] G. Celano, A. Costa, S. Fichera, and G. Perrone. Human factors policy testing in the sequencing on manual mixed-model assembly lines. *Computers & Operations Research*, 31(1):39–59, 2004.

[14] J. A. Díaz and E. Fernández. Hybrid scatter search and path relinking for the capacitated p-median problem. *European Journal of Operational Research*, 169(2):570–585, 2006.

[15] F. Y. Ding, J. Zhu, and H. Sun. Comparing two weighted approaches for sequencing mixed-model assembly lines with multiple objectives. *International Journal of Production Economics*, 102(1):108–131, 2006.

[16] E. Erel, Y. Gocgunz, and I. Sabuncuoğlu. Mixed-model assembly line sequencing using beam search. *International Journal of Production Research*, 45(22):5265–5284, 2007.

[17] C. J. Hyun, Y. Kim, and Y. K. Kim. A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines. *Computers & Operations Research*, 25(7–8):675–690, 1998.

[18] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):265–323, 1999.

[19] H. G. Kim and H. S. Cho. Sequencing in a mixed-model final assembly line with three goals: Simulated annealing approach. *International Journal of Industrial Engineering*, 10(4):607–613, 2003.

[20] T. Korkmazel and S. Meral. Bicriteria sequencing methods for the mixed-model assembly line in just-in-time production systems. *European Journal of Operational Research*, 131(1):188–207, 2001.

[21] 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(23):4955–4974, 2004.

[22] M Laguna and R. Martí. *Scatter Search: Methodology and Implementations in C.* Kluwer, Boston, 2003.

[23] Y. Monden. *Toyota Production System: An Integrated Approach to Just-In-Time.* Chapman & Hall, UK, 3rd edition, 1998.

[24] K. Okamura and H. Yamashina. A heuristic algorithm for the assembly line model-mix sequencing problem to minimize the risk of stopping the conveyor. *International Journal of Production Research*, 17(3):233–247, 1979.

[25] A. Rahimi-Vahed and A. H. Mirzaei. A hybrid multi-objective shuffled frog-leaping algorithm for a mixed-model assembly line sequencing problem. *Computers & Industrial Engineering*, 53(4):642–666, 2007.

[26] P. Ross, S. Schulenburg, J. G. Marín-Blázquez, and E. Hart. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 942–948, San Francisco, 2002. Morgan Kaufmann.

[27] B. R. Sarker and H. Pan. Designing a mixed-model assembly line to minimize the costs of idle and utility times. *Computers & Industrial Engineering*, 34(3):609–628, 1998.

[28] S. Scheuerer and R. Wendolsky. A scatter search heuristic for the capacitated clustering problem. *European Journal of Operational Research*, 169(2):533–547, 2006.

[29] A. Scholl. *Balancing and Sequencing of Assembly Lines*. Springer-Verlag, New York, 1999.

[30] A. Scholl, T. Klein, and W. Domschke. Pattern based vocabulary building for effectively sequencing mixed-model assembly lines. *Journal of Heuristics*, 4(4):359–381, 1998.

[31] R. Tavakkoli-Moghaddam and A. R. Rahimi-Vahed. Multicriteria sequencing problem for a mixed-model assembly line in a JIT production system. *Applied Mathematics and Computation*, 181(2):1471–1481, 2006.

[32] N. T. Thomopoulos. Line balancing – sequencing for mixed model assembly. *Management Science*, 14(2):B59–B75, 1967.

[33] L. H. Tsai. Mixed-model sequencing to minimize utility work and the risk of conveyor stoppage. *Management Science*, 41(3):485–495, 1995.

[34] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[35] Z. Xiaobo and K. Ohno. Algorithms for sequencing mixed models on an assembly line in a JIT production system. *Computers & Industrial Engineering*, 32(1):47–56, 1997.

[36] D. S. Yamashita, V. A. Armentano, and M. Laguna. Scatter search for project scheduling with resource availability cost. *European Journal of Operational Research*, 169(2):623–637, 2004.

[37] C. A. Yano and A. Bolat. Survey, developement, and application of algorithms for sequencing paced assembly lines. *Journal of Manufacturing and Operations Management*, 2:172–198, 1989.

[38] C. A. Yano and R. Rachamadugu. Sequencing to minimize work overload in assembly lines with product options. *Management Science*, 37(5):572–586, 1991.

[39] W. Zeramdini, H. Aigbedo, and Y. Monden. Bicriteria sequencing for just-in-time mixed-model assembly lines. *International Journal of Production Research*, 38(15):3451–3470, 2000.

Appendix

In this appendix the 20 priority rules used in the constructive procedure of the proposed hyper-heuristic are described. This rules are used to select a product between a set of candidates. A product i is a candidate of rule r if $d_i > 0$. Let us the define the following notation.

- I set of products;
- i product i, $i \in I$;
- K set of stations;
- k station k, $k \in K$;
- n_i original demand of product i, $i \in I$;
- t position of the sequence, $(t = 1, \ldots, T)$, $T = \sum_{i=1}^{I} n_i$;
- d_i pending demand of product i (if $t = 0$, $d_i = n_i$), $i \in I$, $t = 1, \ldots, T$;
- p_{ik} processing time for product i in station k, $i \in I$, $k \in K$;
- r_{ik} allocation dynamic index for product i in station k, $r_{ik} = d_i * \Delta_{ik}$, $i \in I$, $k \in K$;
- Δ_{ik} worker displacement in station k due to product i, $\Delta_{ik} = |p_{ik} - c|$, $i \in I$, $k \in K$;
- L_k length of station k, $k \in K$;
- u_i ideal production of product i, $i \in I$;
- s_k initial worker position in station k, $k \in K$;
- w_t accumulated work overload until phase t, $w_t = \sum_1^t w_{tk}$, $k \in K$;
- $\bar{p}$ average processing time $\bar{p} = \sum p_{ik}/(|K| \cdot |I|)$, $i \in I$, $k \in K$;
- k^b bottleneck station;
- w_i work overload produced by product i in current phase t, $w_i = \sum_{k=1}^{K} w_{ti}$, $i \in I$, $k \in K$;
- o_i idle time in the current phase t produced by product i, $o_i = \sum_{k=1}^{K} o_{ti}$, $i \in I$, $k \in K$.

The rules are enumerated as follows:

Rule 1: Select product i^* with largest processing time, $i^* = \arg\max\{p_{ik}\}$.

Rule 2: Select product i^* with smallest processing time, $i^* = \arg\min\{p_{ik}\}$.

Rule 3: Select product i^* with processing time closest to $\bar{p}$, $i^* = \arg\min\{|p_{ik} - \bar{p}|\}$.

Rule 4: Select product i^* with processing time closest to c, $i^* = \arg\min\{\Delta_{ik}\}$.

Rule 5: Select product i^* with largest pending demand, $i^* = \arg\max\{d_i\}$.

Rule 6: Select product i^* with lowest pending demand, $i^* = \arg\min\{p_{ik}\}$.

Rule 7: Select product i^* which produces largest worker displacement in all stations, $i^* = \arg\max\{r_i\}$.

Rule 8: Select product i^* with lowest dynamic index r_{ik}, $\quad i^* = \arg\min\{r_i\}$.

Rule 9: Select product i^* producing the largest total worker displacement in all stations, $\quad i^* = \arg\max\{\sum_{k=1}^{K} \Delta_{ik}\}$.

Rule 10: Select product i^* producing the smallest total worker displacement in all stations, $\quad i^* = \arg\min\{\sum_{k=1}^{K} \Delta_{ik}\}$.

Rule 11: Select product i^* with the largest processing time in the bottleneck station, $\quad i^* = \arg\max\{p_{ik^b}\}$.

Rule 12: Select product i^* with the lowest processing time in the bottleneck station, $\quad i^* = \arg\min\{p_{ik^b}\}$.

Rule 13: Select product i^* closest to the ideal production u_i, $\quad i^* = \arg\min\{|(n_i - d_i) - t \cdot u_i|\}$.

Rule 14: Select product i^* with the largest sum of the dynamic index value for all stations, $i^* = \arg\max \sum_{k=1}^{K} r_{ik}$.

Rule 15: Select product i^* with the smallest sum of the dynamic index value for all stations, $i^* = \arg\min \sum_{k=1}^{K} r_{ik}$.

Rule 16: Select product i^* that produces the largest overload in the current phase, $\quad i^* = \arg\max\{w_i\}$.

Rule 17: Select product i^* that produces the largest idle time in the current phase, $\quad i^* = \arg\max\{o_i\}$.

Rule 18: Select product i^* that produces the smallest overload in the current phase, $\quad i^* = \arg\min\{w_i\}$.

Rule 19: Select product i^* that produces the smallest idle time in the current phase, $\quad i^* = \arg\min\{o_i\}$.

Rule 20: Under the regularity concept, select product i^* producing the work overload value closest to the ideal work overload value (assuming $lbw > 0$), $\quad i^* = \arg\min\{(lbw/T) \cdot t - w_t - w_i\}$; where w_t is the accumulated value.