A NEW APPROACH FOR SOLVING HYBRID FLOWSHOP SCHEDULING PROBLEMS THROUGH EVOLUTIONARY ALGORITHMS.

Fulvio Antonio Cappadonna\textsuperscript{1}, Giovanni Celano\textsuperscript{2}, Antonio Costa\textsuperscript{2}, Sergio Fichera\textsuperscript{2}

\textsuperscript{1}Dep. of Information Technology, Electrical and Electronic Engineering, University of Catania, Italy
\textsuperscript{2}Dep. of Industrial and Mechanical Engineering, University of Catania, Italy,
+39 095 7382423, costa@diim.unict.it

Abstract
This paper addresses a realistic variant of the Hybrid Flow Shop (HFS) problem, based on a real micro-electronics manufacturing environment. Firstly, the formulation of a Mixed Integer Linear Programming (MILP) model for optimally solving the problem is provided. Then, two metaheuristic procedures are presented: the former is a genetic algorithm based on an encoding/decoding method well-known in literature, while the latter combines a genetic algorithm and a proper local search technique exploiting a different encoding/decoding procedure. According to a wide set of instances, the obtained numerical results highlight the effectiveness of the proposed two-phase metaheuristics in terms of both quality of solutions and computational efficiency.

Keywords
Hybrid flowshop, scheduling, genetic algorithms, encoding, decoding.

1. Introduction
Hybrid Flow Shops (HFS) are manufacturing environments that can be found in many practical situations. Differently from traditional flow shop, HFS include several manufacturing stages, instead of a sequence of single machines, wherein each stage holds a set of unrelated parallel machines. As shown by Gupta [1], the problem is NP-hard even if the manufacturing environment is characterized by two stages, the former having two machines and the latter just a single machine. Nevertheless, some mathematical programming approaches have been presented in literature ([2], [3]). The resolution of the HFS problem through the use of metaheuristic techniques represents a common practice in literature, as confirmed by several studies involving both simulated annealing ([4], [5]) and genetic algorithms ([6], [7]). Recently, an exhaustive review on research carried out in the field of HFS problem has been presented by Ruiz and Vázquez-Rodríguez [8].

In this paper, according to a real micro-electronics production environment, a HFS problem including unrelated machines, machine availability constraints, jobs overlapping within the same machine, and limitation on the job waiting time within the inter-stage buffers, is addressed.

The remainder of the paper is organized as follows. In Section 2 the problem statement is reported. In Section 3 a MILP model related to the aforementioned problem is introduced. Section 4 illustrates the two different metaheuristics developed for solving large problem instances. In Section 5 obtained results are reported and discussed. Finally, Section 6 concludes the paper.

2. Problem statement
The proposed HFS problem can be stated as follows. Let us consider a set $I$ of $n$ jobs that has to be worked, without preemption, through $m$ consecutive production stages, aiming at the minimization of the total completion time. The generic, $j$-th stage ($j = 1, 2, \ldots, m$) is composed by $K_j$ unrelated machines. For a given stage, each job has to be processed only by one machine. The set $I$ includes several identical jobs, i.e. jobs having the same processing time; thus, set $I$ may be considered as the union of smaller subsets of identical jobs. Despite the different stages, each machine $k$ ($k = 1, 2, \ldots, K_j$) can work more identical jobs simultaneously (i.e. job overlapping is allowed), according to a given overlapping capacity $N_{jk}$. Due to the maintenance activities, each machine can be subject to unavailability time intervals, being $SL_{ujk}$ and $EU_{ujk}$ the starting time and the ending time of $u$-th unavailability interval of machine $k$ at stage $j$, respectively.
Finally, a time-limit $T_{(j-1)j}$ concerning the permanence of the jobs within the inter-stage buffers, i.e. between stage $j-1$ and stage $j$, is provided.

3. The MILP model

A first goal of the proposed research consists of the development of a Mixed Integer Linear Programming (MILP) model, aiming to optimally solve the aforementioned problem. It is worth pointing out that the proposed MILP model holds several dummy jobs simulating the machine occupancy corresponding to the provided unavailability intervals. Differently from the other jobs, each dummy job can be employed to run one unavailability time-interval for the whole set of machines concerning the multi-stage production system. As for example, if $U (u = 1, 2, \ldots, U)$ unavailability time-intervals have been planned for all the machines, then each $u$-th dummy job will be utilized for simulating the idle time related to a each $u$-th unavailability interval.

Sets and parameters

- $J$: set of stages;
- $K_j$: set of machines at stage $j$;
- $I$: set of real jobs;
- $U$: set of dummy jobs running the unavailability intervals;
- $R = I \cup U$: set of jobs including dummy jobs;
- $S_i$: set of jobs identical to job $i$;
- $T_{(j-1)j}$: maximum waiting time for jobs between stage $j-1$ and stage $j$;
- $SU_{ujk}$: starting time of $u$-th unavailability interval of machine $k$ at stage $j$;
- $EU_{ujk}$: ending time of $u$-th unavailability interval of machine $k$ at stage $j$;
- $D_{ijk}$: processing time of job $i$ on machine $k$ at stage $j$; if $i$ is a dummy job running the $u$-th unavailability interval, $D_{ijk} = EU_{ujk} - SU_{ujk}$;
- $N_{jk}$: maximum number of identical jobs that can be worked simultaneously on machine $k$ at stage $j$;
- $M$: a big number;

Continuous variables

- $X_{ijk}$: starting time of job $i$ on machine $k$ at stage $j$; $i \in R$, $j \in J$, $k \in K_j$
- $C_{max}$: makespan

Binary variables

- $Y_{ijk}$: 1: if job $i$ is worked on machine $k$ at stage $j$ 0; otherwise $i \in R$, $j \in J$, $k \in K_j$
- $W_{ijk}$: 1: if job $i$ and job $l$ are worked simultaneously on machine $k$ at stage $j$ 0 or 1; otherwise $i, l \in I$, $l \in S_i$, $l > i$, $j \in J$, $k \in K_j$
- $Z_{ijk}$: 1: if job $i$ is completed before job $l$ is started on machine $k$ at stage $j$ 0; if job $l$ is completed before job $i$ is started on machine $k$ at stage $j$
- $H_{ijk}$: 1: if job $i$ is completed before job $l$ is started on machine $k$ at stage $j$ 0; if job $i$ starts at a time equal or greater than starting time of job $l$ on machine $k$ at stage $j$ $i, l \in I$, $l \in S_i$, $j \in J$, $k \in K_j$
Q_{iljk} \quad \text{auxiliary variable for either-or constraint} \quad i, l \in I, \quad l > i, \quad j \in J, \quad k \in K_j

\textbf{Model}

\text{minimize } C_{\text{max}}, \text{ subject to:}

\begin{align*}
\sum_{k \in K_j} Y_{jk} &= 1 \quad i \in I, \quad j \in J \quad (1) \\
X_{jk} &\leq M \cdot Y_{jk} \quad i \in I, \quad j \in J, \quad k \in K_j \quad (2) \\
\sum_{k \in K_j} X_{jk} - \sum_{q \in K_j, q \neq j} (X_{ik, j-1q} + D_{ik, j-1q} \cdot Y_{ik, j-1q}) &\geq 0 \quad i \in I, \quad j \in J, \quad j \geq 2 \quad (3) \\
\sum_{k \in K_j} X_{jk} - \sum_{q \in K_j, q \neq j} (X_{ijq} + D_{ijq} \cdot Y_{ijq}) &\leq T_{j-1} \quad i \in I, \quad j \in J, \quad j \geq 2 \quad (4) \\
X_{jk} &= SU_{jk} \quad u \in U, \quad j \in J, \quad k \in K_j \quad (5)
\end{align*}

\begin{align*}
\begin{cases}
X_{jk} + D_{jk} \cdot Y_{jk} \leq X_{jk} + M \cdot Z_{jk} \\
X_{jk} + D_{jk} \cdot Y_{jk} \leq X_{jk} + M \cdot (1 - Z_{jk}) \\
X_{jk} - M \cdot H_{jk} \\
X_{jk} + D_{jk} \cdot Y_{jk} \leq X_{jk} + M \cdot (1 - H_{jk})
\end{cases} \quad i, l \in R, \quad l \in S_j, \quad l > i, \quad j \in J, \quad k \in K_j \quad (6) \\
(1 - W_{jk}) - X_{jk} + X_{jk} + Y_{jk} + Y_{jk} &\leq 2 + M \cdot Q_{jk} \quad i, l \in I, \quad l \in S_j, \quad l > i, \quad j \in J, \quad k \in K_j \quad (7) \\
(1 - W_{jk}) + X_{jk} - X_{jk} + Y_{jk} + Y_{jk} &\leq 2 + M \cdot (1 - Q_{jk}) \quad i, l \in I, \quad l \in S_j, \quad l > i, \quad j \in J, \quad k \in K_j \quad (8)
\end{align*}

\begin{align*}
\sum_{j \in S_k} \sum_{i \in S_i} W_{ijk} &\leq N_{k_i} - 1 \quad i \in I, \quad j \in J, \quad k \in K_j \quad (9) \\
C_{\text{max}} &\geq \sum_{k \in K_j} (X_{i, jk} + D_{i, jk} \cdot Y_{i, jk}) \quad i \in I \quad (10) \\
X_{jk} &\geq 0 \quad i \in I, \quad j \in J, \quad k \in K_j \quad (11)
\end{align*}

Constraint (1) ensures that each real job can be worked by only one machine per stage. Constraint (2) states that if a real job is not assigned to a given machine then its related start processing time must be equal to zero. Constraint (3) ensures that the start processing time of a given job within a given stage must be greater than the completing time of the same job on the previous stage, while constraint (4) shows as the waiting time within the inter-stage buffers must be lower than the fixed time limit. Through constraint (5) a starting time is assigned to each dummy job on each machine, thus being equal to the starting time of the corresponding unavailability interval. Constraints (6) state as two non-identical jobs (i.e. two jobs with different processing times) cannot be overlapped. Constraints (7) handle the possible overlap of identical jobs, i.e. jobs that have the same processing time. Constraints (8) ensure that, if two identical jobs \(i\) and \(l\) are simultaneously worked on machine \(k\) at stage \(j\), variable \(W_{iljk}\) must be equal to one; if jobs \(i\) and \(l\) are not overlapped, then \(W_{iljk}\) indiscriminately could assume value 0 or 1. Constraint (9) handles the capacity limit on the number of identical jobs that can be worked on the same machine. Finally, constraint (10) states that makespan must be higher than the completion time of every job on the last stage \(j\).

\section*{4. Metaheuristic algorithms}

With regards to the optimization techniques adopted to tackle the large-sized problems, metaheuristic procedures performed by Ruiz and Maroto [6] represent the main reference of the present paper. In particular, they deal with the makespan minimization of a \(m\)-stage HFS problem with unrelated parallel machines, sequence dependent setup times and machine eligibility by means of a proper GA equipped with permutation encoding of \(n\) elements, where \(n\) is the number of jobs to be worked.
Whenever a real problem optimization has to be addressed by a metaheuristic algorithm, both the problem representation by a string of digits (i.e. the so-called problem encoding) and the subsequent procedure necessary to evaluate the performance of such a string (i.e. the so-called problem decoding), play a key role under the efficacy and the efficiency viewpoints. The encoding/decoding procedure proposed by Ruiz and Maroto [6] for minimizing the makespan of a HFS problem demonstrated its effectiveness though it is based on a sort of “smart decoding” which cannot investigate the whole space of solutions. The encoding used by the aforementioned authors consists of a permutation string composed by a number of digits equal to the number of jobs to be scheduled, while the related decoding procedure works by assigning, at every stage, each job to the machine which is able to complete such a job earlier than the other machines.

In this paper, a comparison between two distinct optimization procedures has been carried out. The first procedure consists of a genetic algorithm, hereinafter GA_1P, wherein the encoding/decoding approach as defined by Ruiz and Maroto [6] has been properly embedded. Then, in order to enhance the performance of the previous optimization procedure, a new two-phase optimization algorithm, hereinafter GA_2P, able to exploit both a local search procedure and a novel encoding/decoding technique has been developed, thus avoiding any trapping into local optima caused by a “smart approach”.

The proposed GA_1P metaheuristics consists of a genetic algorithm based on the Ruiz and Maroto [6] encoding/decoding scheme, properly adapted to the kind of HFS discussed in this paper. Let us assume to have \( n \) jobs to be worked by a manufacturing system of \( m \) stages; \( K_j \) indicates the number of machines for the generic \( j \)-th stage (\( j = 1, 2, \ldots, m \)). Each string evaluated by the algorithm consists of a permutation \( \pi \) of \( n \) integers. The makespan computation exploits the following procedure. Let us indicate as \( \pi \) the job in the \( i \)-th position of the considered permutation (\( i = 1, 2, \ldots, n \)). With regards to stage \( j \), \( S_{\pi j} \) and \( C_{\pi j} \) represent the starting and the completion times of job \( \pi_i \) at stage \( j \), respectively. For each machine \( k \) pertaining to the stage \( j \) (\( k = 1, 2, \ldots, K_j \)), \( D_{\pi ik} \) is the processing time of job \( \pi_i \), \( L_{jk} \) is the current job worked by the machine, and \( N_{res_{jk}} \) is the residual capacity of the machine (i.e. the residual number of identical jobs that can be still overlapped within the same machine, along with \( L_{jk} \)). The potential starting time \( PS_{\pi ik} \) and potential completion time \( PC_{\pi ik} \) of job \( \pi_i \) on machine \( k \) at the stage \( j \) can be calculated as follows:

\[
PS_{\pi ik} = \begin{cases} S_{\pi k} & \text{if } \pi_i \text{ and } L_{jk} \text{ are identical, } C_{\pi_{(j-1)}} \leq S_{\pi k} \text{ and } N_{res_{jk}} > 0 \\ \max \{C_{\pi k}; C_{\pi_{(j-1)}}\} & \text{otherwise} \end{cases}
\]

\[
PC_{\pi ik} = PS_{\pi ik} + D_{\pi ik}.
\]

Once these quantities are computed, a check concerning the unavailability intervals involving the same machine has to be performed with the aim of avoiding any interference between the job processing time and the planned unavailability time interval. Whether such an interference exists, it has to be:

\[
PS_{\pi ik} = EU_{u_{jk}},
\]

\[
PC_{\pi ik} = PS_{\pi ik} + D_{\pi ik},
\]

where \( EU_{u_{jk}} \) represents the completion time of the unavailability interval \( u \)-th, concerning machine \( k \) at the stage \( j \). Finally, once the set of potential completion times for job \( \pi_i \) at stage \( j \), and for all machines have been computed, the effective completion time is calculated as follows:

\[
C_{\pi j} = \min_{k=1}^{K_j} \{PC_{\pi ik}\}.
\]

Due to the recursive approach characterizing the present decoding procedure, the start processing time of job \( \pi_i \) can be subsequently computed by:

\[
S_{\pi j} = C_{\pi j} - D_{\pi_{j,k}},
\]

where \( k \) represents the selected machine of stage \( j \) able to minimize the job \( \pi_i \) completion time.

Finally, the makespan can be computed as follows:
Once the decoding procedure has been applied to the whole sequence of jobs to be processed and once the makespan has been computed, the waiting time limit constraint concerning the inter-stage buffers should be verified, thus avoiding any unfeasible solution. The proposed decoding procedure cope with the waiting time constraint by both a repair algorithm and a penalty function. The repair algorithm tries to reduce the level of unfeasibility of a solution by postponing as much as possible the release time of all the jobs into the first stage. Whether the repairing procedure does not allow any unfeasibility reduction, a proper penalty function is employed to increase the fitness of the unfeasible solution.

With regards to GA_1P perturbation operators, a position based crossover has been employed, while a simple gene swapping technique has been used for mutation. An elitism procedure has also been adopted, in order to save, at each generation, the best individuals of the population. In order to enhance the ability of the metaheuristics in investigating a wider space of solutions, a multi-stage encoding has then been developed. According to this approach, instead of using a unique permutation string for all the production stages, a number of permutation strings equal to the number of stages have been considered for the problem encoding. The decoding procedure used by GA_1P can be easily adapted to the proposed multi-stage encoding which, in addition, does not provide any modification to both the penalty function and the repairing algorithm, necessary to manage the waiting time constraint.

Since the implementation of a genetic algorithm entirely based on this multi-stage encoding has led to poor results, both in terms of quality of solutions and computational burden, a two-phase metaheuristics GA_2P has been developed. The first phase of the proposed optimization algorithm is carried out through the application of the single permutation-encoding GA_1P. After a defined number of iterations is reached, the best obtained solution (i.e. the permutation string related to the best individual) is re-coded in the multi-stage encoding, just by repeating \( m \) times the corresponding job sequence. At the same time the second phase of the metaheuristics can be executed. In fact, as the new multi-stage encoding can explore a wider set of solutions, despite a higher computational burden due to the new string, an agile local search-based algorithm has been implemented. The adopted local search can be classified as a random search technique wherein the current solution is replaced only by a new solution able to improve the reached objective function value. The perturbation mechanism is based on the shift insertion method but, if the neighborhood of the selected job is populated by other identical jobs, then the entire group of jobs is shifted.

5. Numerical examples and computational results

In order to assess performances of developed algorithms, a set of 40 instances has been generated on the basis of a production system characterized by 3 stages and 5 machines per stage. Four different sized classes of problems have been considered on the basis of the number of jobs, namely \( n = \{10, 30, 50, 100\} \). Each class held 10 instances and for each instance, 5 replications with different random seeds were considered. Processing times were generated in the range \([1, 99]\). Each machine had a number of unavailability intervals drawn from an uniform distribution in the range \([0, 2]\), whose starting and ending times were randomly picked in the range \([1, 99* (n/5)]\) and then sorted in ascending order. The size of each subset of identical jobs was lower than 5, while the capacity of each machine, namely the maximum number of jobs to be simultaneously worked, was generated by uniform distribution in the range \([1, 5]\). Finally, maximum waiting times in inter-stage buffers were randomly extracted from the range \([50, 150]\).

With reference to GA_1P, a population size \( P_{size} = 100 \) was chosen, with a mutation probability \( p_{mut} = 0.05 \) and an adaptive crossover probability \( p_{cross} = \exp (-1/G_i) \), where \( k \) is the current iteration and \( G_i \) is the residual number of generations, until the stopping criterion (fixed at 50,000 makespan evaluations) is reached. As concerns the proposed two-phase metaheuristics GA_2P, the first phase exit criterion has been set equal to 33,300 objective evaluations while the second phase fulfils the local search procedure along with the multi-stage encoding for 16,700 evaluations.

With reference to the class I (\( n = 10 \)), the global optimum was found by the MILP model executed on an IBM ILOG CPLEX® Vers. 12.2 (64 bit) platform. For the same class of problem, Table 1 shows on the
left the average CPU time required by each metaheuristics while, on the right, the average percentage gap from the global optimum has been reported.

Table 1. Average performances of GA_1P and GA_2P for Class I \((n = 10)\).

<table>
<thead>
<tr>
<th>Class</th>
<th>Average CPU time (s)</th>
<th>Average percentage gap from global optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>I ((n = 10))</td>
<td>GA_1P 282.79</td>
<td>GA_2P 282.10</td>
</tr>
</tbody>
</table>

As concerns the large sized problems, Table 2 shows the average CPU time required by both GA_1P and GA_2P, and the makespan average percentage reduction obtained by GA_2P in respect to GA_1P.

Table 2. Average performances of GA_1P and GA_2P for Classes II, III, IV \((n = 30, 50, 100)\).

<table>
<thead>
<tr>
<th>Class</th>
<th>Average CPU time (s)</th>
<th>Average makespan reduction in best solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>II ((n = 30))</td>
<td>GA_1P 767.46</td>
<td>GA_2P 806.06</td>
</tr>
<tr>
<td>III ((n = 50))</td>
<td>GA_1P 1,637.07</td>
<td>GA_2P 1,757.79</td>
</tr>
<tr>
<td>IV ((n = 100))</td>
<td>GA_1P 3,785.27</td>
<td>GA_2P 4,150.51</td>
</tr>
</tbody>
</table>

6. Conclusions

In this paper, a HFS problem based on a real micro-electronics manufacturing environment has been addressed with regards to the makespan minimization objective. A MILP model has been developed in order to assess the performance of two different metaheuristic procedures: the former being a conventional GA based on a permutation encoding inspired to a well-known literature work, the second being a two-phase metaheuristics which starts as the previous GA and then switches to a multi-stage encoding embedded into a proper local search algorithm. Results obtained show how the two-phase proposed metaheuristics leads to more effective outcomes, systematically reaching the global optimum in small sized instances solved by the MILP model. In respect to large sized problem, the proposed approach confirms its efficacy both in terms of quality of solutions and computational burden, in spite of a slight increase in the CPU times.

References