UNIVERSITY OF PADUA

ENGINEERING FACULTY

DEPARTMENT OF MANAGEMENT AND ENGINEERING

Bachelor degree

ANALYSIS OF THE PRODUCTION SCHEDULING METHODS INTO HYBRID FLOW SHOP ENVIRONMENTS

ANALISI DEI METODI DI SCHEDULAZIONE DELLA PRODUZIONE IN AMBIENTI HYBRID FLOW SHOP

Supervisor: Prof. Faccio Maurizio

Co-supervisor: Eng. Zanin Giorgia

Student: Vitale Niccolò

ACADEMIC YEAR 2010/2011

Index

Summary

The objective of the work is to deepen into hybrid flow shop environments and scheduling methods, in order to define theoretical aspects, and the practical ones with particular attention for the relationship with the course made by prof. Faccio named “Impianti Industriali”.
Basically it consists in a first phase of research into the literature, and then a second phase of critical analysis, and deepen of related concepts.
To make the research remarkable a little number of papers have been considered, exactly thirty, because a bigger number could be too much to be analyzed properly; and, more importantly, only recent papers, because firstly they represent the present state of art, and secondly into their bibliography other relevant papers can be found, helping the reader to understand the subjects treated, and to deepen it if he wants.
The literature analysis should highlight instruments, methods, algorithms judged interesting and possible object for application and evaluation.
The second part of the work is a critical analysis of these papers, and the contents, trying to understand advantages and eventual errors or problems or risks related to these concepts.
Finally, after fully understand all of this, a little elaboration is proposed, with the goal of signal the direction took from the scientists and showed by the literature, and maybe some personal opinions.

Chapter 1:
Introduction to scheduling in hybrid flow shop environments

1.1 The scheduling problem

Scheduling is an important tool for manufacturing and engineering, where it can have a major impact on the productivity of a process. In manufacturing, the purpose of scheduling is to minimize the production time and costs, by telling a production facility when to make, with which staff, and on which equipment. Production scheduling aims to maximize the efficiency of the operation and reduce costs.

Production scheduling tools greatly outperform older manual scheduling methods. These provide the production scheduler with powerful graphical interfaces which can be used to visually optimize real-time work loads in various stages of production, and pattern recognition allows the software to automatically create scheduling opportunities which might not be apparent without this view into the data. For example, an airline might wish to minimize the number of airport gates required for its aircraft, in order to reduce costs, and scheduling software can allow the planners to see how this can be done, by analyzing time tables, aircraft usage, or the flow of passengers.

Companies use backward and forward scheduling to allocate plant and machinery resources, plan human resources, plan production processes and purchase materials.

Forward scheduling is planning the tasks from the date resources become available to determine the shipping date or the due date.

Backward scheduling is planning the tasks from the due date or required-by date to determine the start date and/or any changes in capacity required.

The benefits of production scheduling include: process change-over reduction, inventory reduction, reduced scheduling effort, increased production efficiency, real time information.

Various charts are used to help schedulers visually manage schedules and constraints. The Gantt chart (Pinedo, 2002) is a display that shows activities on a horizontal bar graph in which the bars represent the time of the activity. Below is an example of a Gantt chart.

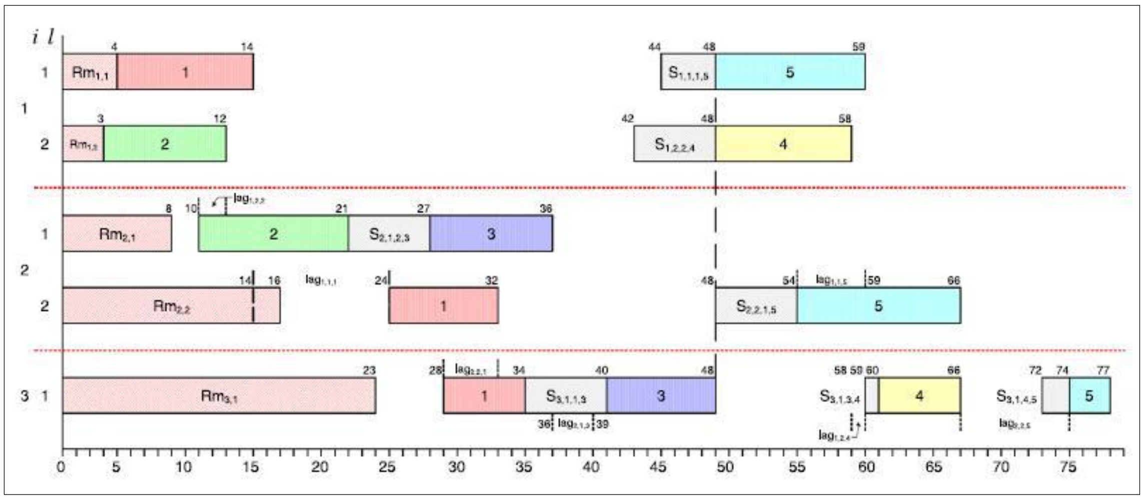


Fig 1.1: Gantt-chart for an example problem, Ruiz et al. (2008)

1.1.1 NP-hardness and the travelling salesman problem

NP-hard (non-deterministic polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP". A problem H is NP-hard if and only if there is an NP-complete problem L that is polynomial time Turing-reducible (from Alan Turing, it means that the L problem is solvable knowing H problem solution) to H. In other words, L can be solved in polynomial time by an oracle machine with an oracle for H. Informally, we can think of an algorithm that can call such an oracle machine as a subroutine for solving H, and solves L in polynomial time, if the subroutine call takes only one step to compute. NP-hard problems may be of any type: decision problems, search problems, or optimization problems (Garey & Johnson, 1979). As consequences of definition, we have (note that these are claims, not definitions):

- Problem H is at least as hard as L, because H can be used to solve L;
- Since NP-complete problems transform to each other by polynomial-time many-one reduction (also called polynomial transformation), all NP-complete problems can be solved in polynomial time by a

reduction to H, thus all problems in NP reduce to H; note, however, that this involves combining two different transformations: from NP-complete decision problems to NP-complete problem L by polynomial transformation, and from L to H by polynomial Turing reduction;

- If an optimization problem H has an NP-complete decision version L, then H is NP-hard.

A common mistake is to think that the NP in NP-hard stands for non-polynomial. Although it is widely suspected that there are no polynomial-time algorithms for NP-hard problems, this has never been proven. Moreover, the class NP also contains all problems which can be solved in polynomial time.

An historical example of NP-hard problem is the travelling salesman problem *(TSP)*: Given a list of cities and their pair-wise distances, the task is to find a shortest possible tour that visits each city exactly once.

The problem was first formulated as a mathematical problem in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, a large number of heuristics and exact methods are known, so that some instances with tens of thousands of cities can be solved.The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing. In these applications, the concept city represents, for example, customers, soldering points, or DNA fragments, and the concept distance represents travelling times or cost, or a similarity measure between DNA fragments. In many applications, additional constraints such as limited resources or time windows make the problem considerably harder.

In the theory of computational complexity, the decision version of the TSP belongs to the class of NP-complete problems. Thus, it is likely that the worst case running time for any algorithm for the TSP increases exponentially with the number of cities.

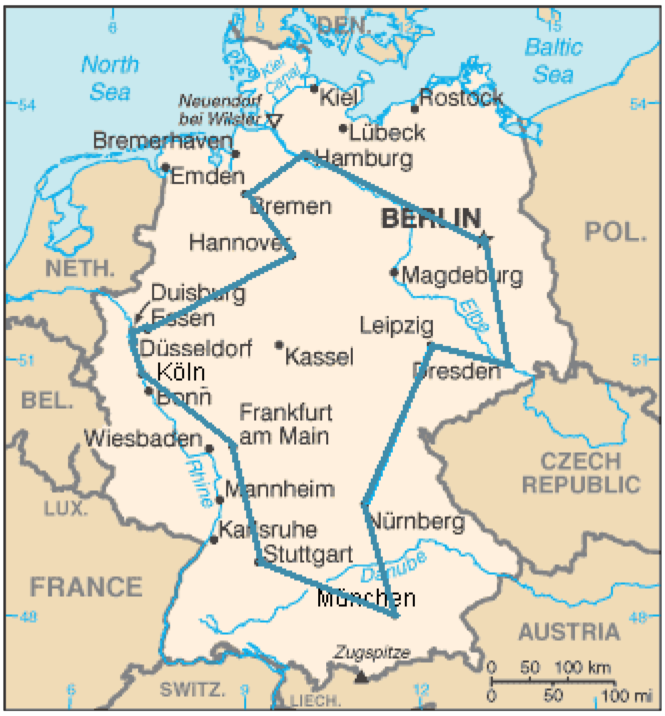


Fig. 1. 2: An optimal TSP tour through Germany's 15 largest cities. It is the shortest among 43589145600 (14!/2) possible tours visiting each city exactly once, Central Intelligence Agency (2007)

1.2 Hybrid Flow Shop environments

A Hybrid Flow Shop *(HFS)* consists of series of production stages, each of which has several machines operating in parallel, that can be identical, uniform or unrelated. Some stages may have only one machine, but at least one stage must have multiple machines. The flow of jobs through the shop is unidirectional. Each job is processed by one machine in each stage and it must go through one or more stage (Elmaghraby & Kamoub, 1997).

A job consists of several operations to be performed by none, one or more machines on each stage. The *i*-th operation of a job *i*, to be performed at the *i*-th stage, requires units of time and can start only after the completion of the previous operation from the operation sequence of this job. This definition is very general and is the basis of the papers reviewed (Ribas et al. 2010).

The HFS problem can also be represented as a graph G(N,A), where N is a set of nodes corresponding to each operation, and A is a set of disjunctive arcs describing the set of possible paths in the graph. A solution is a graph G(N, S), where S is a subset of the arcs in A but with a fixed direction, i.e., S represents an assignment and ordering of the job operations. Several heuristics have been devised using these representation (Ruiz & Vazquez-Rodriguez, 2010).

1.2.1 Identical, uniform and unrelated machines

In the standard situation, the machines are totally identical, despite that each job may have its own processing criteria on the machine.
When the machines are not identical it should be defined that, they can be considered uniform if the speed in which they execute the jobs is uniform, i.e. machines power do not depend on the jobs they are running. In other words, if p_{ij} is the processing time (or weight) J_i on machine M_j, and $p_{ij'}$ is the processing time on machine $M_{j'}$. For Job $J_{i'}$, $p_{i'j}$ is the processing time on machine M_j and $p_{i'j'}$ is the processing time on machine $M_{j'}$, then $p_{ij}=p_{i'j}=p_{ij'}=p_{i'j'}$.
If the parallel machines are non-uniform they are called unrelated and and they execute jobs in accordance with their power, of course their speed is non uniform.
These distinction is very important because the problem of scheduling is really different in these cases, especially in the last one.

1.2.2 Batch production

Batch production is the manufacturing technique of creating a group of components at a workstation before moving the group to the next step in production.
There are several advantages of batch production; it can reduce initial capital outlay because a single production line can be used to produce several products. As shown in the example, batch production can be useful for small businesses who cannot afford to run continuous production lines. If a retailer buys a batch of a product that does not sell, then the producer can cease production without having to sustain huge losses.
Batch production is also useful for a factory that makes seasonal items, products for which it is difficult to forecast demand, a trial run for production, or products that have a high profit margin. Batch production

also has disadvantages. There are inefficiencies associated with batch production as equipment must be stopped, re-configured, and its output tested before the next batch can be produced. Idle time between batches is known as downtime. The time between consecutive batches is known as cycle time. Cycle time variation is a Lean Manufacturing metric.

1.2.3 Buffer between stages

Between the various stages of an hybrid flow shop environment there can be a definite amount of space, called buffer. This can be a part of the conveyor belt, or a disarticulation in the flow, or even a defined space out of the same flow. This is a location in where the workflow can wait to be worked in the next stage; the dimension of the buffer is an index of efficiency and scheduling precision.
The papers with a limited buffer dimension will be indicated into the table, because of their importance in the HFS scheduling future applications. It’s obvious that the dimension of the buffer gives an amount to the waiting time of the jobs, and also to the waiting jobs quantity. These is told because into the papers you can find the same thing (buffer) with different names.

1.3 Model of paper

Usually, an old model has been well tested and used for a long time so it is secure that it is capable to represent the real manufacturing system. Using old models is the easiest way in which to evaluate the performance but it has a low level of innovation. Developing existing old models is a better practice because it make possible to improve the efficiency of the model, usually taking into consideration factors that were not been taken into account previously.
New models are the most innovative and they usually evaluate aspects never studied before.
The aggregation of existing models is a good approach to have an overall manufacturing performance. Since each kind of model fits well only a group of issues, it’s difficult to use only one type of method and take into account all the problems.

1.4 Mathematical algorithms

To solve a difficult problem, such as hybrid flow shop scheduling, a lot of very different algorithms have been developed. Some of them are heuristics, other are metaheuristics, and other are exact; the most important and significant ones are here presented, and also a definition of these three terms.

1.4.1 Heuristic algorithms

The term heuristic is used for algorithms which find solutions among all possible ones, but they do not guarantee that the best will be found, therefore they may be considered as approximately and not accurate algorithms. These algorithms, usually find a solution close to the best one and they find it fast and easily. Sometimes these algorithms can be accurate, that is they actually find the best solution, but the algorithm is still called heuristic until this best solution is proven to be the best. The method used from a heuristic algorithm is one of the known methods, such as greediness, but in order to be easy and fast the algorithm ignores or even suppresses some of the problem's demands.

An example for the TSP is here given: Let's number the cities *from 1 to n* and let city 1 be the city-base of the salesman. Also let's assume that *c(i,j)* is the visiting cost *from i to j*. There can be *c(i,j)* different from *c(j,i).* pparently all the possible solutions are *(n-1)!*. Someone could probably determine them systematically, find the cost for each and everyone of these solutions and finally keep the one with the minimum cost. These requires at least *(n-1)!* steps.

If for example there were 21 cities the steps required are *(n-1)! = (21-1)!=20!* steps. If every step required a *msec* we would need about 770 centuries of calculations. Apparently,the exhausting examination of all possible solutions is out of the question. Since we are not aware of any other quick algorithm that finds a best solution we will use a heuristic algorithm. According to this algorithm whenever the salesman is in town *i* he chooses as his next city, the city *j* for which the *c(i,j)* cost, is the minimum among all *c(i,k)* costs, where *k* are the pointers of the city the

salesman has not visited yet. There is also a simple rule just in case more than one cities give the minimum cost, for example in such a case the city with the smaller *k* will be chosen. This is a greedy algorithm which selects in every step the cheapest visit and does not care whether this will lead to a wrong result or not.
This example shows how an heuristic is implemented, but also why an heuristic is used: when a solution is not required to be optimal, but just good, and you don't want to lose a lot of time with exact algorithms, an heuristic method is the correct way to do. A lot of heuristics are used, in fact, to reduce computational time, in hybrid with other more accurate algorithms.

1.4.2 Metaheuristic algorithms

In computer science, metaheuristic designates a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Metaheuristics make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics do not guarantee an optimal solution is ever found. Many metaheuristics implement some form of stochastic optimization.
Metaheuristics are used for combinatorial optimization in which an optimal solution is sought over a discrete search-space. An example problem is the travelling salesman problem where the search-space of candidate solutions grows more than exponentially as the size of the problem increases which makes an exhaustive search for the optimal solution infeasible. This phenomenon is commonly known as the curse of dimensionality.
Popular metaheuristics for combinatorial problems include genetic algorithms by Holland (1975) ant colony optimization by Dorigo (1992), simulated annealing by Kirkpatrick et al. (1983).
These three are all present in some of the reviewed papers, but a lot of other are also present. A definition of the given algorithms is here given, according to a general view some algorithms are unified into nature inspired category, because of their origins, and most of all because their authors were "nature inspired" when they created it.

1.4.3 Genetic algorithms

The genetic algorithm *(GA)* mimics the process of natural evolution. This metaheuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms *(EA)*, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

In a genetic algorithm, a population of strings (called chromosomes or the genotype of the genome), which encode candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem, evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Once we have the genetic representation and the fitness function defined, GA proceeds to initialize a population of solutions randomly, then improve it through repetitive application of mutation, crossover, inversion and selection operators.

Initialization: initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, covering the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

Selection: during each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.

Most functions are stochastic and designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large, preventing premature convergence on poor solutions. Popular and well-studied selection methods include roulette wheel selection and tournament selection.

Reproduction: the next step is to generate a second generation population of solutions from those selected through genetic operators: crossover (also called recombination), and/or mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions, for reasons already mentioned above.

Termination: this generational process is repeated until a termination condition has been reached. Common terminating conditions are:- a solution is found that satisfies minimum criteria; - fixed number of generations reached; - allocated budget (computation time/money) reached; - the highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results; - manual inspection; -combinations of the above.

1.4.4 Ant colony optimization

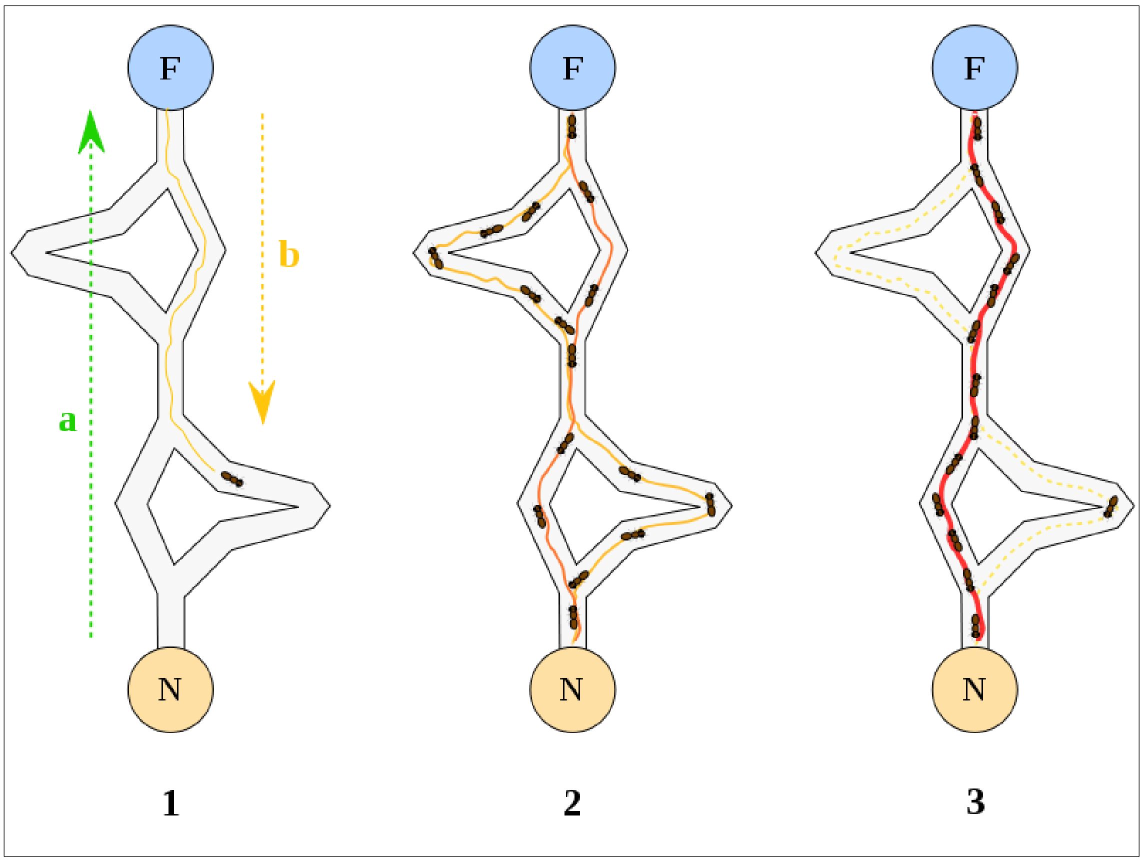


Fig. 1.3: Ant colony optimization: 1) The first ant find a food source (F), Using some path (a), then it comes back to the nest (N), laying a pheromone trail. 2) the ants follow one of the 4 possible paths, but the reinforcement of the trail make the shortest path more appealing. 3) the ants follow the shortest path, the pheromone trail of the longest ones evaporate, Dréo, (2006)

Ant colony optimization *(ACO)* is a metaheuristic algorithm inspired from exploitation of food resources among ants. In the natural world, ants (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but to instead follow the trail, returning and reinforcing it if they eventually find food.

Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over faster, and thus the pheromone density remains high as it is laid on the path as fast as it can evaporate. Pheromone evaporation has also the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained.

Thus, when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads all the ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve:

1. The first ant finds the food source (F), via any way (a), then returns to the nest (N), leaving behind a trail pheromone (b) *(see image)*
2. Ants indiscriminately follow four possible ways, but the strengthening of the runway makes it more attractive as the shortest route.
3. Ants take the shortest route, long portions of other ways lose their trail pheromones.

In a series of experiments on a colony of ants with a choice between two unequal length paths leading to a source of food, biologists have observed that ants tended to use the shortest route (Goss et al., 1989; Deneuburg et al. 1990). A model explaining this behaviour is as follows:

1. An ant (called "blitz") runs more or less at random around the colony;
2. If it discovers a food source, it returns more or less directly to the nest, leaving in its path a trail of pheromone;
3. These pheromones are attractive, nearby ants will be inclined to follow, more or less directly, the track;
4. Returning to the colony, these ants will strengthen the route;
5. If there are two routes to reach the same food source then, in a given amount of time, the shorter one will be traveled by more ants than the long route;
6. The short route will be increasingly enhanced, and therefore become more attractive;
7. The long route will eventually disappear because pheromones are volatile;
8. Eventually, all the ants have determined and therefore "chosen" the shortest route.

Ants use the environment as a medium of communication. They exchange information indirectly by depositing pheromones, all detailing the status of their "work". The information exchanged has a local scope, only an ant located where the pheromones were left has a notion of them. The mechanism to solve a problem too complex to be addressed by single ants is a good example of a self-organized system. This system is based on positive feedback (the deposit of pheromone attracts other ants that will strengthen it themselves) and negative (dissipation of the route by evaporation prevents the system from thrashing). Theoretically, if the quantity of pheromone remained the same over time on all edges, no route would be chosen. However, because of feedback, a slight variation on an edge will be amplified and thus allow the choice of an edge. The algorithm will move from an unstable state in which no edge is stronger than another, to a stable state where the route is composed of the strongest edges.
The basic philosophy of the algorithm involves the movement of a colony of ants through the different states of the problem influenced by two local decision policies, *trails* and *attractiveness*. Thereby, each such ant incrementally constructs a solution to the problem. When an ant completes a solution, or during the construction phase, the ant evaluates the solution and modifies the trail value on the components used in its solution. This pheromone information will direct the search of the future ants. Furthermore, the algorithm also includes two more mechanisms, viz., *trail evaporation* and *daemon actions*. *Trail evaporation* reduces all trail values over time thereby avoiding any possibilities of getting stuck in local optima. The *daemon actions* are used to bias the search process from a non-local perspective.

1.4.5 Simulated annealing

Simulated annealing *(SA)* is a generic probabilistic metaheuristic for the global optimization problem of applied mathematics, namely locating a good approximation to the global optimum of a given function in a large search space.
The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of

its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.
By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends both on the difference between the corresponding function values and also on a global parameter *T* (called the temperature), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when *T* is large, but increasingly "downhill" as *T* goes to zero. The allowance for "uphill" moves saves the method from becoming stuck at local optima—which are the bane of greedier methods.
In the simulated annealing *(SA)* method, each point s of the search space is analogous to a state of some physical system, and the function *E(s)* to be minimized is analogous to the internal energy of the system in that state. The goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy.
The basic iteration: at each step, the SA heuristic considers some neighboring state *s'* of the current state *s*, and probabilistically decides between moving the system to state *s'* or staying in state *s*. These probabilities ultimately lead the system to move to states of lower energy. Typically this step is repeated until the system reaches a state that is good enough for the application, or until a given computation budget has been exhausted.
The neighbors of a state: the neighbours of a state are new states of the problem that are produced after altering the given state in some particular way. For example, in the traveling salesman problem, each state is typically defined as a particular permutation of the cities to be visited. The neighbours of some particular permutation are the permutations that are produced for example by interchanging a pair of adjacent cities. The action taken to alter the solution in order to find neighbouring solutions is called "move" and different "moves" give different neighbours. These moves usually result in minimal alterations of the solution, as the previous example

depicts, in order to help an algorithm to optimize the solution to the maximum extent and also to retain the already optimum parts of the solution and affect only the suboptimum parts. In the previous example, the parts of the solution are the parts of the tour.
Searching for neighbors to a state is fundamental to optimization because the final solution will come after a tour of successive neighbors. Simple heuristics move by finding best neighbor after best neighbor and stop when they have reached a solution which has no neighbors that are better solutions. The problem with this approach is that a solution that does not have any immediate neighbors that are better solution is not necessarily the optimum. It would be the optimum if it was shown that any kind of alteration of the solution does not give a better solution and not just a particular kind of alteration. For this reason it is said that simple heuristics can only reach local optima and not the global optimum. Metaheuristics, although they also optimize through the neighborhood approach, differ from heuristics in that they can move through neighbors that are worse solutions than the current solution. Simulated Annealing in particular doesn't even try to find the best neighbor. The reason for this is that the search can no longer stop in a local optimum and in theory, if the metaheuristic can run for an infinite amount of time, the global optimum will be found.
Acceptance probabilities: The probability of making the transition from the current state s to a candidate new state s' is specified by an acceptance probability function *P(e,e',T),* that depends on the energies *e = E(s)* and *e' = E(s')* of the two states, and on a global time-varying parameter T called the temperature. One essential requirement for the probability function *P* is that it must be *nonzero* when *e' > e*, meaning that the system may move to the new state even when it is worse (has a higher energy) than the current one. It is this feature that prevents the method from becoming stuck in a local minimum, a state that is worse than the global minimum, yet better than any of its neighbors.

1.4.6 Exact algorithms

Approximate solutions are not always allowed or wanted, sometimes even if a very long computational time is needed, exact algorithms are used. The principal advantage of these algorithms is that they find for sure an optimum solution. There are a lot of exact algorithms, the two we decided to define are: branch and bound algorithms and linear programming.

1.4.7 Branch and bound algorithms

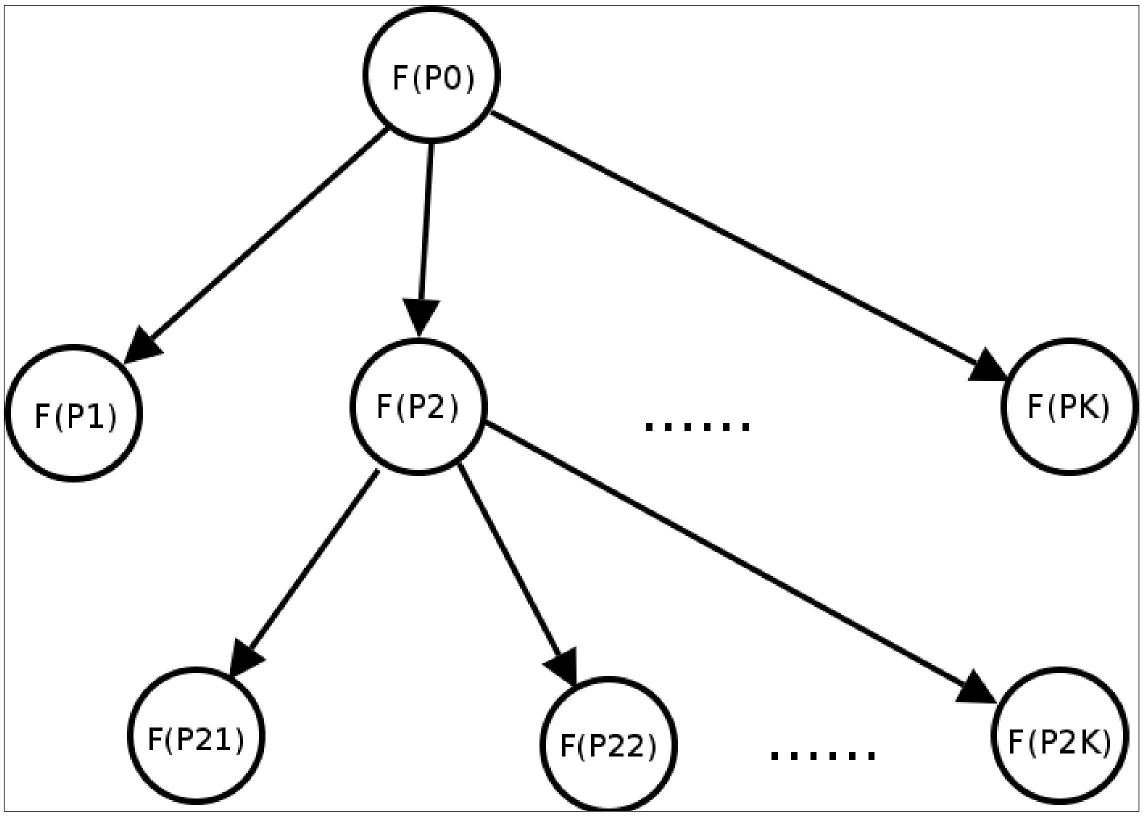


Fig. 1.4: Schematic representation of the scomposition in sub-problems used in branch and bound, Strocchi (2006)

Branch and bound (*BB* or *B&B*) is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization. It consists of a systematic enumeration of all candidate solutions, where large subsets of fruitless candidates are discarded en masse, by using upper and lower estimated bounds of the quantity being optimized. The method was first proposed by Land & Doig (1960) for discrete programming.

For definiteness, we assume that the goal is to find the minimum value of a function *f(x),* where *x* ranges over some set *S* of admissible or candidate solutions (the search space or feasible region). Note that one can find the maximum value of *f(x)* by finding the minimum of *g(x) = −f(x).* (For example, *S* could be the set of all possible trip schedules for a bus fleet, and *f(x)* could be the expected revenue for schedule *x*.)

A branch-and-bound procedure requires two tools. The first one is a splitting procedure that, given a set *S* of candidates, returns two or more smaller sets $S_1, S_2, \dots$ whose union covers *S*. Note that the minimum of *f(x)* over *S* is $min\{v_1, v_2, \dots\}$, where each v_i is the minimum of *f(x)* within S_i. This step is called branching, since its recursive application defines a tree structure (the search tree) whose nodes are the subsets of *S*.

Another tool is a procedure that computes upper and lower bounds for the minimum value of *f(x)* within a given subset *S*. This step is called bounding.

The key idea of the BB algorithm is: if the lower bound for some tree node (set of candidates) A is greater than the upper bound for some other node B, then A may be safely discarded from the search. This step is called pruning, and is usually implemented by maintaining a global variable *m* (shared among all nodes of the tree) that records the minimum upper bound seen among all sub-regions examined so far. Any node whose lower bound is greater than *m* can be discarded.

The recursion stops when the current candidate set *S* is reduced to a single element; or also when the upper bound for set *S* matches the lower bound. Either way, any element of *S* will be a minimum of the function within *S*.

The efficiency of the method depends strongly on the node-splitting procedure and on the upper and lower bound estimators. All other things being equal, it is best to choose a splitting method that provides non-overlapping subsets.

Ideally the procedure stops when all nodes of the search tree are either pruned or solved. At that point, all non-pruned sub-regions will have their upper and lower bounds equal to the global minimum of the function. In practice the procedure is often terminated after a given time; at that point, the maximum lower bound and the minimum upper bound, among all non-pruned sections, define a range of values that contains the global minimum. Alternatively, within an overriding time constraint, the algorithm may be terminated when some error criterion, such as $(max - min)/(min + max)$, falls below a specified value.

The efficiency of the method depends critically on the effectiveness of the branching and bounding algorithms used; bad choices could lead to

repeated branching, without any pruning, until the sub-regions become very small. In that case the method would be reduced to an exhaustive enumeration of the domain, which is often impractically large. There is no universal bounding algorithm that works for all problems, and there is little hope that one will ever be found; therefore the general paradigm needs to be implemented separately for each application, with branching and bounding algorithms that are specially designed for it.
Branch and bound methods may be classified according to the bounding methods and according to the ways of creating/inspecting the search tree nodes.

1.4.8 Linear programming

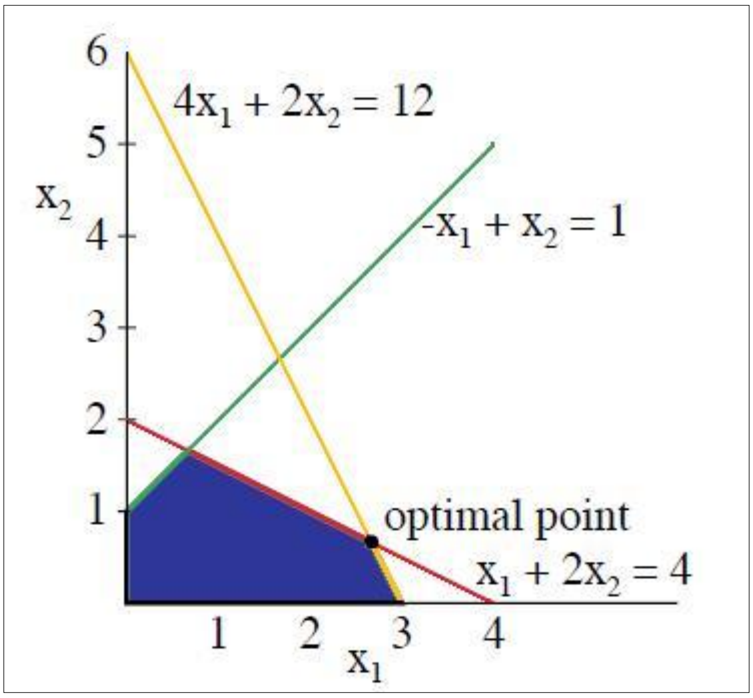


Fig. 1.5: A diagram showing an example of a linear programming problem and the way in which the feasible region is bounded by straight, Ferguson (2000)

Linear programming *(LP)* is a mathematical method for determining a way to achieve the best outcome (such as maximum profit or lowest cost) in a given mathematical model for some list of requirements represented as linear relationships.
More formally, linear programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality

constraints. Given a polytope and a real-valued affine function defined on this polytope, a linear programming method will find a point on the polytope where this function has the smallest (or largest) value if such point exists, by searching through the polytope vertices.
Linear programs are problems that can be expressed in canonical form: *Maximize* $c^T x$ *; Subject to* $A x \leq b$*.* Where x represents the vector of variables (to be determined), c and b are vectors of (known) coefficients and A is a (known) matrix of coefficients. The expression to be maximized or minimized is called the objective function ($c^T x$ in this case). The equations $A x \leq b$ are the constraints which specify a convex polytope over which the objective function is to be optimized. (In this context, two vectors are comparable when every entry in one is less-than or equal-to the corresponding entry in the other. Otherwise, they are incomparable.)
Linear programming can be applied to various fields of study. It is used most extensively in business and economics, but can also be utilized for some engineering problems. Industries that use linear programming models include transportation, energy, telecommunications, and manufacturing. It has proved useful in modeling diverse types of problems in planning, routing, scheduling, assignment, and design.

1.4.9 Hybrid algorithms

A lot of algorithms have been developed in the last century. An intelligent procedure to approach the HFS problem is to use the best part of some algorithm fuse together with best part of other algorithms. There are some algorithms that are quick and easy to use, some other maybe slow and difficult to implement, an intelligent hybridization of these algorithms can give optimum solutions in less time or maybe better solutions if the two divided algorithms give just good solutions. Another presented approach of the hybridization is to divide the population into sub-populations and apply the algorithm firstly to every sub-population, this is a method used to reduce the number of individuals to control, and to avoid the curse of dimensionality.

1.4.10 Reduction of computational time

We have seen that some algorithms could employ a large amount of computational time to solve, this can be repaired or enveloped. When an algorithms is employed just to reduce the computational time, and it is assumable that the proposed method could work and reach optimum solution even without this algorithm, it's interesting to type this; into the 2.1 table this is showed with a [C] into the correct algorithm. Of course no exact algorithms can be used to reduce computational time, but heuristic or metaheuristic, because they approximate some procedure or sub-applications that aren't the most important part of the method, the one that brings to the optimum solution, but are very slow to compute.

1.5 Analysis of results

In the manufacturing environment, there are a lot of factors and issues to be evaluated. In the papers seen typically two factors have been evaluated: makespan and tardiness, with the goal of minimizing them. These factors and problems sometimes have different nature and cannot be evaluated with the same tool. Since the model that allow to measure the performance should try to fit in the best way the problem and the factor to evaluate, they are available different types of models. Hence, they have to be chosen according to the nature of the system.

1.5.1 Parameters to be minimized: makespan and tardiness

Into the reviewed papers a lot of parameters have been used as the most important to evaluate, and most of all, to minimize. Makespan, or, in other words, total completion time, is the one minimized into the bigger part of the seen papers, and we can be sure of that seeing Ruiz & Vazquez-Rodriguez (2010), where it's typed that the 60% of the papers review by the two authors have the object of minimizing total completion time, or makespan (C_{max}). Other parameters are connected with the makespan, such as total/average (weighted) completion time for example. Seeing this analogy between this factors into the 2.1 table we assumed that this was the same parameter, but it’s just a simplification to make the table smaller. Another diffuse parameter to minimize is the tardiness, into a lot of papers the goal was to minimize the maximum tardiness, but there are other similar factors such as total/average (weighted) tardiness, and into the 2.1 table we do the same thing as makespan, for the same reason.

1.5.2 Statistical indexes

Using statistical method it is possible to improve the quality of data with the design of experiments and survey sampling. Statistics also provides tools for prediction and forecasting using data and statistical models. Statistics is applicable to a wide variety of industrial measurement. Statistical methods can be used to summarize or describe a collection of data; this is called descriptive statistics. This is useful in research, when communicating the results of experiments. In addition, patterns in the data may be modelled in a way that accounts for randomness and uncertainty in the observations, and are then used to draw inferences about the process or population being studied; this is called inferential statistics. Inference is a vital element of scientific advance, since it provides a prediction (based in data) for where a theory logically leads. To further prove the guiding theory, these predictions are tested as well, as part of the scientific method. If the inference holds true, then the descriptive statistics of the new data increase the soundness of that hypothesis.
New researches are trying to develop classical statistical methods in order to better suit the real environment and application in modern manufacturing environment, overcoming the old frameworks used in the past that no more suit the new applications.
Into the 2.1 table the papers where clear statistical indexes have been used are signalled.

1.5.3 Dispatching rules as a parameter of comparison

Dispatching rules are specific parameters that define how to give priority to the works. These are the most important dispatching rules:

- *SPT (Shortest Processing Time):* Highest priority is given to the waiting operation with the shortest imminent operation time.
- *LPT (Longest Processing Time):* Highest priority is given to the waiting operation with the longest imminent operation time.
- *MWKR (Most Work Remaining):* Highest priority is given to the waiting operation associated with the job having the most total processing time remaining to be done.
- *LWKR (Least Work Remaining):* Highest priority is given to the waiting operation associated with the job having the least amount of total processing time remaining to be done.
- *TWORK (Total Work):* Highest priority is given to the job with the least total processing requirement on all operations.
- *FIFO (First In First Out):* Highest priority is given to the waiting operation that arrived at the queue first.
- *LIFO (Last In First Out):* Highest priority is given to the waiting operation that arrived at the queue last (Dominic et al. 2004).

These are not directly considerable as a parameter of comparison, but into the reviewed papers it was found that, the results where compared using the different dispatching rules, to show a more accurate and complete view of the results, and doing that, dispatching rules are in fact used to evaluate an algorithm or a method against another.

Chapter 2: **Survey of the production scheduling methods into hybrid flow shop environments**

2.1 Classification table

Thirty papers have been read and summarized. It is very difficult to explain how the works are similar and why the same works are different, so the simpler way to show this is a table, with a short number of factors, such as: model, development, problem statements, mathematical algorithms, results and comparation. These factors have been chosen after a long work into larger and more detailed other tables in order to give the better one to the reader.

				study			statements					mathematical algorithm								results			comparison						
				model	development		machine		batch	buffer	heuristic	genetic	nature inspired	simulate annealing	branch & bound	linear programming	other	hybrid	parameter/s		index/es	algorithm						
	paper	authors	year		static	dynamic	single	multi											makespan	tardiness		HE	GA	nature	SA	BB	LP	other
1	*A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setu*	Mirsanei et al.	2010	D	X		X	I		L				X					X		S		X	X				
2	*An efficient bi-objective heuristic for scheduling of hybrid flow shops*	Mousavi et al.	2010	N	X			I			X								X	X	R				X			X
3	*Hybrid flow-shop: a memetic algorithm using constraint-based scheduling for efficient search*	Jouglet et al.	2009	A	X			I				X			X			X	X		S		X			X		
4	*A genetic algorithm for solving no-wait flexible flow lines with due window and job rejection*	Jolai et al.	2008	D	X			I		N		X				X		X			S		X				X	X
5	*A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environm*	Tang et al.	2005	N		X		I			C		X						X	X	R							
6	*Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdow*	Gholami et al.	2008	D		X	X	I	X	U		X							X		S							
7	*Using ant colony optimization to solve hybrid flow shop scheduling problems*	Alaykyran et al.	2007	D	X		X	I					X						X							X		
8	*An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrela*	Rashidi et al.	2010	N	X			D		N		X					C	X	X	X			X					
9	*A class of hypothesis-test-based genetic algorithms for flow shop scheduling with stochastic processing*	Wang et al.	2005	N	X		X					X							X		S		X					
10	*Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum*	Naderi et al.	2008	D	X			I						X			X	X	X	X	S	X	X	X				
11	*A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan*	Wang et al.	2010	D	X			I						X					X		S		X	X				
12	*Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteri*	Jungwattanakit et al.	2007	A	X			D			X	X				X			X	X	R							
13	*Sequencing and scheduling of nonuniform flow pattern in parallel hybrid flow shop*	Rathinasamy & R	2009	N		X		I	X	U	X								X		R							
14	*Optimal scheduling of a two-stage hybrid flow shop*	Haouari et al.	2006	D	X			I			X				X			X	X									
15	*A genetic algorithm for hybrid flow shop scheduling with multiprocessor tasks*	Oguz & Erkan	2005	D	X		X	I			X	X						X	X		S							X
16	*Evaluating virtual cells and multistage flow shops: an analytical approach*	Vakharia et al.	2000	N	X			I	X	N									X									X
17	*A multi-objective particle swarm for a flow shop scheduling problem*	Rahimi-Vahed et al.	2006	D		X		I		N			X				X	X	X	X	S		X					
18	*Scheduling algorithms to minimize the number of tardy jobs in two-stage hybrid flow shops*	Choi & Lee	2009	N	X			I		U	C				X					X		X						
19	*Hybrid flow shop scheduling with parallel batching*	Amin-Naseri & Beheshti-Nia	2009	N		X		I	X	L	X	X			X				X			X						
20	*A meta-heuristic approach to solve a JIT scheduling problem in hybrid flow shop*	Khalouli et al.	2010	D		X		I			X		X							X								
21	*An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times*	Zandieh et al.	2006	D	X			I					X						X		S		X					
22	*A case study in a two-stage hybrid flow shop with setup time and dedicated machines*	Lin & Jiao	2003	D	X			D	X		X									X	R					X		
23	*Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requiremen*	Voß & Witt	2005	A		X		I	X	L	X									X	R							
24	*A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems*	Zandieh et al.	2009	N	X			D		U		X							X			X						
25	*An effective hybrid de-based algorithm for multi-objective flow shop scheduling with limited buffers*	Qian et al.	2007	A		X	X			L		X	X					X	X				X					
26	*Multiprocessor task scheduling in multistage hybrid flow-shops: a parallel greedy algorithm approach*	Kahraman et al.	2010	N		X		I									X		X		S		X					
27	*A note on the two-stage assembly flow shop scheduling problem with uniform parallel machines*	Koulamas & Kyparisis	2007	N	X		X	U			X								X									
28	*Tsp-based scheduling in a batch-wise hybrid flow-shop*	Caricato et al.	2005	N	X			I	X	N		X							X		S	X					X	
29	*Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its applicati*	Choi et al.	2010	N		X		I		L							X		X	X								
30	*An Exact Approach for Batch Scheduling in Flexible Flow Lines with Limited Intermediate Buffers*	Sawik	2002	D	X			I	X	L						X			X									

2.1.1 Legend

Model: [A] is for aggregation, it means that the paper uses a lot of other previous works and puts them together. [N] is for new, it means that the paper presents a new solution, or a new environment, for example, it doesn't means that it's totally new, but that being very different to the previous works it can be considered new. [D] is for development, it means that a previous work have been developed, maybe adapting it to other new or different environments, the difference between a new and a developed paper is that a developed paper is more influenced from the previous papers.

Machine: In the multi-machine environment there are three factors, based on the characteristics of the parallel machines. [I] is for identical machines. [U] is for uniform machines. [D] is for unrelated or different or maybe dedicated machines.

Problem statements: When a buffer between stages is considered, a distinction is given, based on buffer dimension. [N] is for no buffer, or very small buffer. [L] is for limited buffer. [U] is for unlimited buffer.

Mathematical algorithms: Into the introduction chapter the computational time reduction problem is presented. Into some papers heuristics or metaheuristics algorithms are given just for reducing computational time. When this happens the algorithms is evidenced with [C].

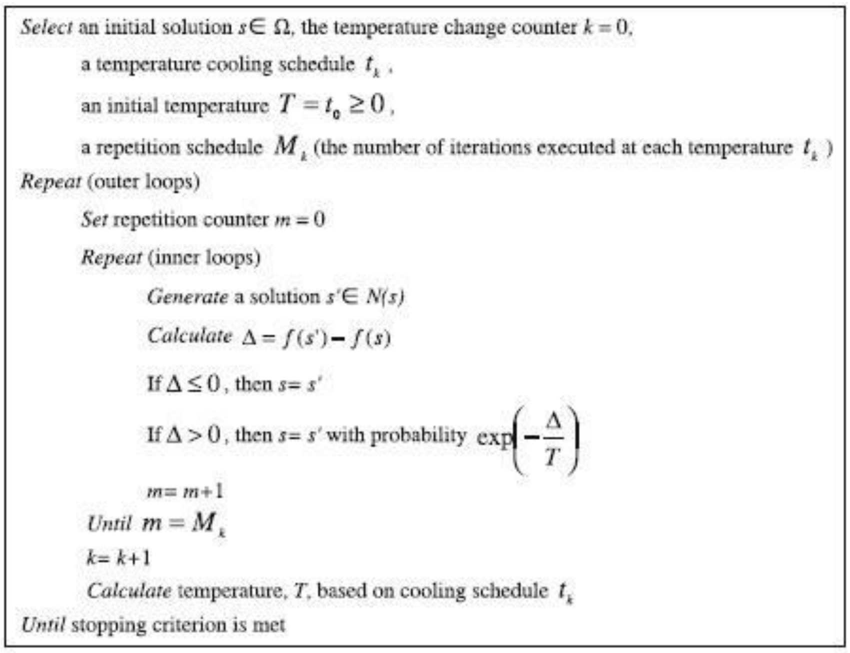
Results: When showing the results, some papers uses index or indexes to evaluate, sometimes these are common statistical indexes such as average or average percent deviation, for example; other times these index are dispatching rules, it is not correct to call these rules indexes, but the use the authors give sometimes of it is similar and comparable with statistical indexes. [S] is for statistical indexes. [R] is for dispatching rules.

Note that for more about the legend and the parameters can be found in the introduction chapter.

2.2 Reviewed and summarized papers

Following there are the summarized papers. The publications have been selected in order to have recent and valuable researches, there are no papers released before 2000. For each one of them is presented a short summary, that presents the problem, and the solution algorithm/s given from the authors. In order to respect the authors works the articles have been shorted but the order and scope have been respected. A selection of interesting pictures took from the papers is also present.

2.2.1 H. S. Mirsanei, M. Zandieh, M. J. Moayed, M. R. Khabbazi; *A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times.* JOURNAL OF INTELLIGENT MANUFACTURING, 18 January 2010 (DOI 10.1007/s10845-009-0373-8)



```
Select an initial solution $s \in \Omega$, the temperature change counter $k = 0$,
    a temperature cooling schedule $t_k$,
    an initial temperature $T = t_0 \geq 0$,
    a repetition schedule $M_k$ (the number of iterations executed at each temperature $t_k$)
Repeat (outer loops)
    Set repetition counter $m = 0$
    Repeat (inner loops)
        Generate a solution $s' \in N(s)$
        Calculate $\Delta = f(s') - f(s)$
        If $\Delta \leq 0$, then $s = s'$
        If $\Delta > 0$, then $s = s'$ with probability $\exp\left(-\frac{\Delta}{T}\right)$
        $m = m+1$
    Until $m = M_k$
    $k = k+1$
    Calculate temperature, $T$, based on cooling schedule $t_k$
Until stopping criterion is met
```

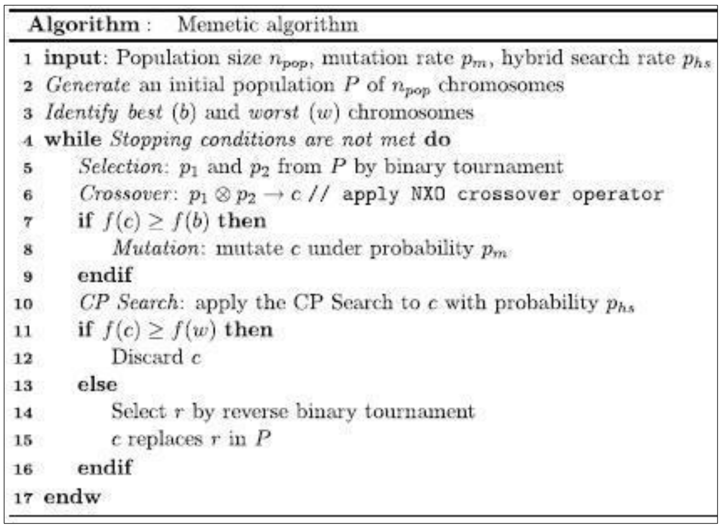
Fig. 2.1: The pseudo-code of Simulated Annealing, Mirsanei et al. (2009)

In most real industries such as chemical, textile, metallurgical, printed circuit board, and automobile manufacturing, hybrid flow shop problems have seq.-dependent setup times (*SDST*). In this research, the problem of SDST hybrid flow shop scheduling with parallel identical machines to minimize the makespan is studied. A novel simulated annealing (*NSA*) algorithm is developed to produce a reasonable manufacturing schedule within an acceptable computational time. In this study, the proposed NSA uses a well combination of two moving operators and a SPT Cyclic Heuristic of Kurz & Askin (2004) for generating new solutions. The obtained results are compared with those computed by Random Key Genetic Algorithm (RKGA) purposed by Kurz & Askin (2004) and with those computed by Immune Algorithm (IA) purposed by Zandieh et al. (2006) which are proposed previously, using graphics and tables, based on the relative percent deviation (RPG) and the average computational time. The results show that NSA outperforms both RKGA and IA in almost every case.

2.2.2 S. M. Mousavi, M. Zandieh, M. Amiri; *An efficient bi-objective heuristic for scheduling of hybrid flow shops.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, 27 October 2010 (DOI 10.1007/s00170-010-2930-x)

This paper considers the problem of scheduling *n* independent jobs in hybrid flow shop environment with sequence-dependent setup times to minimize the makespan and total tardiness. For the optimization problem, an algorithm namely bi-objective heuristic (BOH) is proposed for searching Pareto-optimal frontier. The term is named after Vilfredo Pareto, an Italian economist who used the concept in his studies of economic efficiency and income distribution. Given an initial allocation of goods among a set of individuals, a change to a different allocation that makes at least one individual better off without making any other individual worse off is called a Pareto improvement. An allocation is defined as "Pareto optimal" when no further Pareto improvements can be made. The aim of the proposed algorithm is to generate a good approximation of the set of efficient solutions. The BOH procedure initiates by generating a seed sequence. Since the output results are strongly dependent on the initial solution and in order to increase the quality of output results algorithm, it is considered how the generation of seed sequence with random way and particular sequencing rules. Two methods named Euclidean distance and percent error have been proposed to compare non-dominated solution sets obtain of each seed sequence. It is perceived from these methods that the generation of seed sequence using earliest due date rule is more effective. Then, the performance of the proposed BOH is compared with a simulated annealing (MOSA) purposed by Loukil et al. (2007) and a variable neighborhood search (VNS) heuristic purposed by Prandtstetter & Raidl (2007) on a set of test problems. The data envelopment analysis is used to evaluate the performance of approximation methods. From the results obtained, it can be seen that the proposed algorithm is efficient and effective.

2.2.3 A. Jouglet, C. Oguz, M. Sevaux; *Hybrid Flow-Shop: a memetic algorithm using constraint-based scheduling for efficient search.* JOURNAL OF MATHEMATICAL MODELLING AND ALGORITHMS, (2009) 8: 271–292



```
Algorithm :  Memetic algorithm
input: Population size $n_{pop}$, mutation rate $p_m$, hybrid search rate $p_{hs}$
Generate an initial population $P$ of $n_{pop}$ chromosomes
Identify best ($b$) and worst ($w$) chromosomes
while Stopping conditions are not met do
    Selection: $p_1$ and $p_2$ from $P$ by binary tournament
    Crossover: $p_1 \otimes p_2 \rightarrow c$ // apply NXO crossover operator
    if $f(c) \geq f(b)$ then
        Mutation: mutate $c$ under probability $p_m$
    endif
    CP Search: apply the CP Search to $c$ with probability $p_{hs}$
    if $f(c) \geq f(w)$ then
        Discard $c$
    else
        Select $r$ by reverse binary tournament
        $c$ replaces $r$ in $P$
    endif
endw
```

Fig. 2.2: Jouglet et al. (2009)

The paper considers the hybrid flow-shop scheduling problem with multiprocessor tasks. Motivated by the computational complexity of the problem, it is proposed a memetic algorithm for this problem in the paper. First of all they describe the implementation details of a genetic algorithm, which is used in the memetic algorithm. Then they propose a constraint programming based branch-and-bound algorithm to be employed as the local search engine of the memetic algorithm. In other words memetical algorithm is a combination of genetical and branch-and-bound algorithms the same hybridization purposed by Portmann et al. (1998) . They then explain the computational experiments carried out to evaluate the performance of three algorithms (genetic algorithm of Oguz et al. (2004), constraint programming based branch-and-bound algorithm of Portmann et al. (1998) , and memetic algorithm) in terms of both the quality of the solutions produced and the efficiency, explaining it better they compare the single methods with the hybridization which the memetic algorithm uses. These results demonstrate that this algorithm produces better solutions and that it is very efficient, also because it can be used in many flow shop typical problems such as $F_k(Pm_1, \ldots, Pm_k)|size_{ij}|C_{max}$ and $F_k(Pm_1, \ldots, Pm_k)||C_{max}$ using the classification of Ribas et al (2009) .

2.2.4 F. Jolai, S. Sheikh, M. Rabbani, B. Karimi; *A genetic algorithm for solving no-wait flexible flow lines with due window and job rejection.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, (2009) 42: 523–532

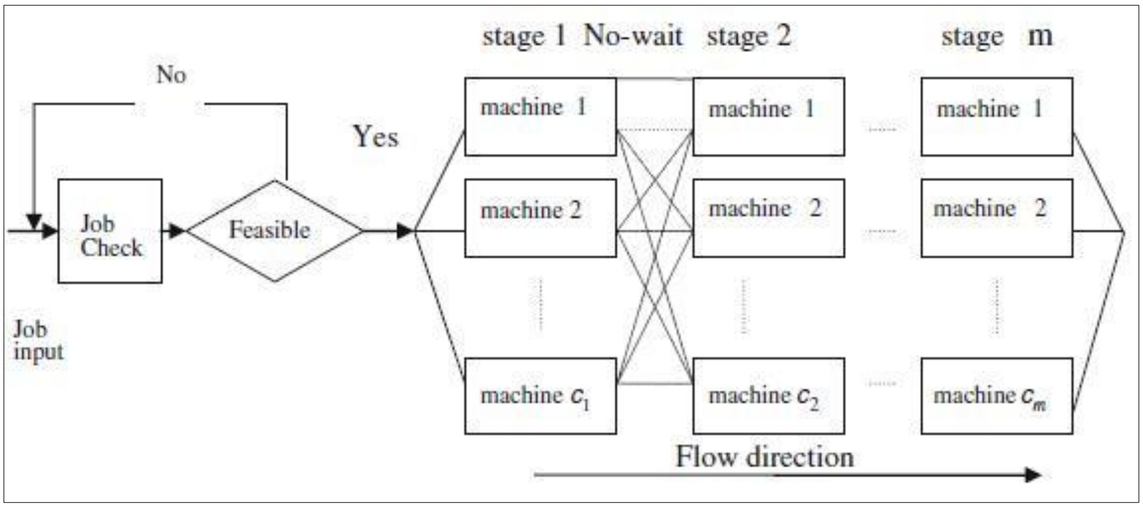


Fig. 2.3: Schematic of no-wait FSMP, Joulai et al.(2009)

This paper addresses a no-wait multi-stage flexible flow shop problem. Note that some jobs may be rejected. A mixed integer linear programming model with the objective of maximizing the total profit gained from scheduled jobs is introduced. Since the problem is NP-hard and difficult to find an optimal solution in a reasonable computational time, an efficient genetic algorithm is presented as the solution procedure. A heuristic mechanism is proposed to use in each generation of the genetic algorithms to assure the feasibility and superior quality of the obtained solutions. This paper develop the production and delivery scheduling problem with time windows (PDPTW), the improved method is then compared with one of the previous solutions introduced by Garcia & Lozano (2005). Computational results show that the presented approach performs considerably in terms of both quality of solutions and required runtimes. However there are places in industry where a fixed time lag between stages is mandatory. This situation may arise where inspection or transportation is an integral part of production. This leads to extension of this model to a more general case where a fixed time (greater or equal 0) enters between machine operations in subsequent stages, to adapt the model to a diffused situation.

2.2.5 L. Tang, W. Liu, J. Liu; *A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment.* JOURNAL OF INTELLIGENT MANUFACTURING, (2005) 16: 361–370

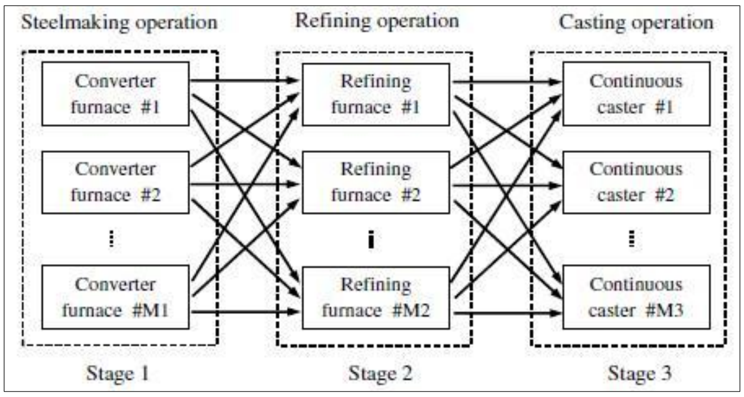


Fig. 2.4: A pratical example of dynamic HFS, Tang et al. (2005)

HFS is fairly common in flexible manufacturing and in process industry. Because manufacturing systems often operate in a stochastic and dynamic environment, dynamic hybrid flow shop scheduling is frequently encountered in practice. This paper proposes a neural network model and algorithm to solve the dynamic hybrid flow shop scheduling problem. Neural networks (NN) are collections of mathematical models that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning. The key element of the NN paradigm is the novel structure of the information processing system. The advantage of NN lies in their resilience against distortions in the input data and their capability of learning. In order to obtain training examples for the neural network, they first study, through simulation, the performance of some dispatching rules that have demonstrated effectiveness in the previous related researches from Liu & Dong (1996). The results are then transformed into training examples. The training process is optimized by the delta-bar-delta (DBD) method that can speed up training convergence, all of this is showed with graphics and tables. The most commonly used dispatching rules are used as benchmarks. Simulation results show that the performance of the neural network approach is much better than that of the traditional dispatching rules. The authors of the paper type that there can be some other types of random events in practical dynamic production environment, such as machine breakdown, rush orders and order cancellation etc. Further research is needed to develop methods for problems with such events.

2.2.6 M. Gholami, M. Zandieh, A. Alem-Tabriz; *Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, (2009) 42: 189–201

Pinedo (1995) cited that machine setup time is a significant factor for production scheduling in all flow patterns, and it may easily consume more than 20% of available machine capacity if not well handled. In addition, the completion time of production and machine setups are influenced by production mix and production sequence. On the one hand, processing in large batches may increase machine utilization and reduce the total setup time but would increase the flow time. There is a trade-off between flow time and machine utilization by selecting batch size and scheduling. Scheduling problems with sequence-dependent setup times are among the most difficult classes of scheduling problems. This paper deals with the hybrid flow shop scheduling problems in which there are sequence-dependent setup times, commonly known as the SDST, and machines which suffer stochastic breakdown to optimize objectives based on expected makespan. This type of production system is found in industries such as chemical, textile, metallurgical, printed circuit board, and automobile manufacture. With the increase in manufacturing complexity, conventional scheduling techniques for generating a reasonable manufacturing schedule have become ineffective. The genetic algorithm can be used to tackle complex problems and produce a reasonable manufacturing schedule within an acceptable time. This paper describes how we can incorporate simulation into genetic algorithm approach to the scheduling of a SDST hybrid flow shop with machines that suffer stochastic breakdown. An overview of the hybrid flow shops and scheduling under stochastic unavailability of machines are presented. The results obtained with genetic algorithm approach are analyzed with Taguchi experimental design. The "parameter design" developed by Dr. Taguchi in early 1960s can be applied to process design. Generally, parameter design procedures can be explained as follows: 1. The influences of controllable factors on the S/N ratio and mean of the response are evaluated. In fact, they test the appropriate experimental design on S/N ratio and mean of the considered characteristic. 2. For each factor which has significant impact on the S/N ratio, the level which increases the S/N ratio will be selected. 3. Each factor which does not have any significant impact on S/N ratio, and has significant impact on mean of the response (y), is considered as adjustment factor, and the level whose mean of y is closer to objective point will be selected. 4. Factors which have significant impact neither on S/N ratio nor on mean of y, are taken into consideration as economical factors, and the levels that decrease cost of production will be selected. Authors state that future studies can focus on the other features of breakdown events such as non-resumable case and other distributions for both time between failures (MTBF) and time to repair (MTTR). In addition, an experimental design considering interaction effects among factors for a further study may be devised.

2.2.7 K. Alaykýran, O. Engin, A. Döyen; *Using ant colony optimization to solve hybrid flow shop scheduling problems.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, (2007) 35: 541–550

Ant colony optimization (ACO) is a new and encouraging group of "nature inspired" algorithms. The ant system (AS) is the first algorithm of ACO. In this study, an improved ACO method is used to solve hybrid flow shop (HFS) problems. This algorithm is used into a lot of problems, it is particularly adaptable to every graph optimal route problem, the authors improved and adapt these methods to the HFS problem. The operating parameters of AS have an important role on the quality of the solution. In order to achieve better results, a parameter optimization study is conducted in this paper. In order to evaluate the success of the algorithm on HFS problems, it was been run on 63 different benchmark problems taken from Carlier & Neron (2000), the same as those used Neron et al. (2001), and Engin & Doyen (2004). At the end of the study, there is a comparison of the performance : the results show that the improved ACO method is an effective and efficient method for solving HFS problems. In authors opinion the inspiration of nature in problem solving seems to be increasing its impact on researchers in future due to its encouraging performance and adaptability to other problems. Better results may be achieved by utilizing hybrid or parallel applications and also be fine tuning the parameters of these problem solving methods. The proposed method may be used in hybrid with other metaheuristics, such as genetic algorithms or simulated annealing.

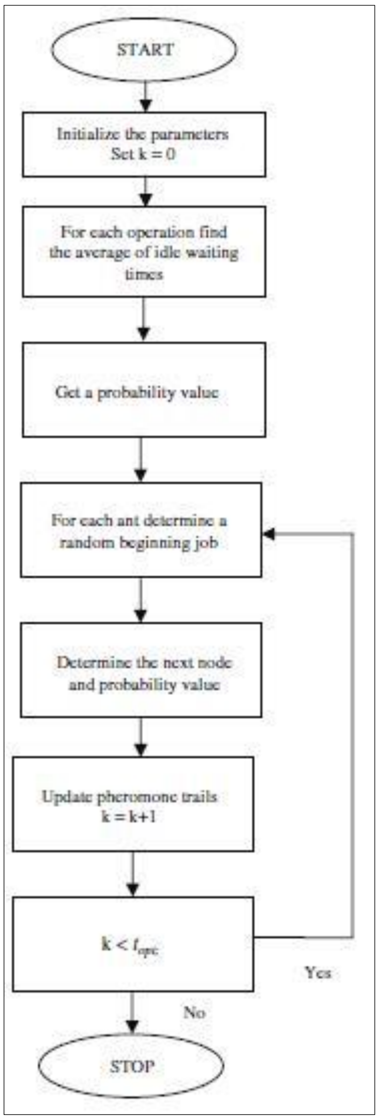

START
Initialize the parameters
Set k = 0
For each operation find the average of idle waiting times
Get a probability value
For each ant determine a random beginning job
Determine the next node and probability value
Update pheromone trails
k = k+1
$k < t_{opt}$
Yes
No
STOP

Fig. 2.5: Flow chart for the improved ACO, Alaykyran et al. (2007)

2.2.8 E. Rashidi, M. Jahandar, M. Zandieh; *An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, (2010) 49: 1129–1139

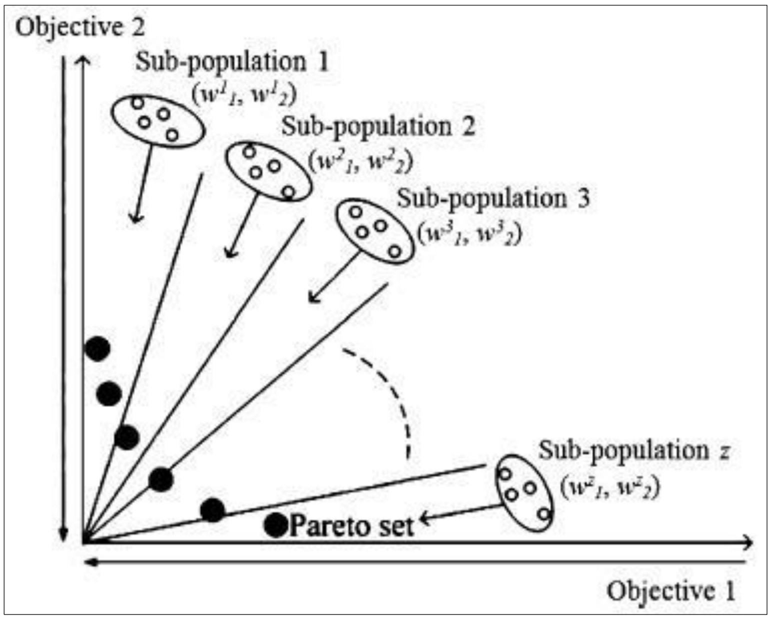


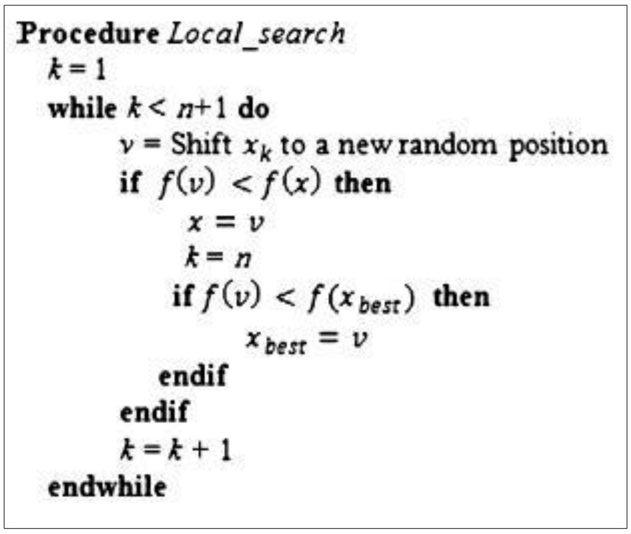
Fig. 2.6: The whole population is divided into several subpopulations, Chang et al. (2007)

In this paper, the authors address the hybrid flow shop scheduling problems with unrelated parallel machines, sequence-dependent setup times and processor blocking to minimize the makespan and maximum tardiness, so this can be called a bi-criteria. They type that this should be the first work that try to solve this hard situation. Since the problem is strongly NP-hard, they propose an effective algorithm consisting of independent parallel genetic algorithms by dividing the whole population into multiple subpopulations. Each subpopulation will be assigned with different weights to search for optimal solutions in different directions. To further cover the Pareto solutions, each algorithm is combined with a novel local search step and a new helpful procedure called Redirect. The proposed Redirect procedure tries to help the algorithm to overcome the local optimums and to further search the solution space. When a population stalls over a local optimum, at first, the algorithm tries to achieve a better solution by implementing the local search step based on elite chromosomes. As implementing the local search step is time-consuming, it's proposed a method to speed up the searching procedure and to further increase its efficiency. If the local search step failed to work, then the Redirect procedure changes the direction and refreshes the population. Computational experiments indicate that this improving procedures are thriving in achieving better solutions. The obtained results are interesting for the proposed algorithm considering both measures chosen. They purpose some improvements to the method such as, adding limited buffers to the situation, or working with more than two criteria.

2.2.9 L. Wang, L. Zhang, D.-Z. Zheng; *A class of hypothesis-test-based genetic algorithms for flow shop scheduling with stochastic processing time.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, (2005) 25: 1157–1163

This paper, studies a bit different problem, such as the flow shop with stochastic processing time. It has been decided that a different situation could be interesting to evaluate the solutions elaborated by genetic algorithm in other scheduling problems. As an important optimization problem with a strong engineering background, stochastic flow shop scheduling with uncertain processing time is difficult because of inaccurate objective estimation, huge search space, and multiple local minima, especially NP-hardness. As an effective meta-heuristic, genetic algorithms (GAs) have been widely studied and applied in scheduling fields, but so far seldom for stochastic cases. In this paper, a hypothesis-test method, an effective methodology in statistics, is employed and incorporated into a GA to solve the stochastic flow shop scheduling problem and to avoid premature convergence of the GA. The proposed approach is based on statistical performance and a hypothesis test. It not only preserves the global search ability of a GA, but it can also reduce repeated searches for those solutions with similar performance in a statistical sense so as to enhance population diversity and achieve better results. Simulation results based on some benchmarks from Pinedo (1995) demonstrate the feasibility and effectiveness of the proposed method by comparison with a traditional GA purposed by Goldberg (1989). The effects of some parameters on the performance of the proposed algorithms are also discussed.

2.2.10 B. Naderi, M. Zandieh, V. Roshanaei; *Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness.* . THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, (2009) 41:1186–1198



```
Procedure Local_search
  $k = 1$
  while $k < n+1$ do
        $v$ = Shift $x_k$ to a new random position
        if $f(v) < f(x)$ then
            $x = v$
            $k = n$
            if $f(v) < f(x_{best})$ then
                $x_{best} = v$
            endif
        endif
        $k = k + 1$
  endwhile
```

Fig. 2.7: The procedure of the local search, Naderi et al. (2009)

This article addresses the problem of scheduling hybrid flowshops where the setup times are sequence dependent to minimize makespan and maximum tardiness. To solve such an NP-hard problem, they introduce a novel simulated annealing (SA) with a new concept, called "Migration mechanism", and a new operator, called "Giant leap", to bolster the competitive performance of SA through striking a compromise between the lengths of neighborhood search structures. The authors hybridize the SA (HSA) with a simple local search to further equip the algorithm with a new strong tool to promote the quality of final solution of SA. The procedure of this local search is described as follows: The first job in the sequence *x*, called *x_1* is relocated into a new random position in the sequence *v*. If this new sequence *v* results in better objective function, the current solution *x* is replaced by the new sequence *v*. This procedure iterates at most for all the subsequent jobs in the sequence *x,* all of this is showed in *fig. 6*. Two basically different objective functions, minimization of makespan and maximum tardiness, are taken into consideration to evaluate the robustness and effectiveness of the HSA. Furthermore, they explore the effects of the increase in the number of jobs on the performance of the algorithm in terms of both the acceptability of the solution quality and robustness.

2.2.11 H.-M. Wang, F.-D. Chu, F.-C. Wu; *A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, 28 July 2010 (DOI 10.1007/s00170-010-2868-z)

This paper studies a hybrid flow shop scheduling problem (HFS) with multiprocessor tasks, in which a set of independent jobs with distinct processor requirements and processing times must be processed in a k-stage flow shop to minimize the makespan criterion. This problem is known to be NP-hard, thus providing a challenging area for meta-heuristic approaches. This paper develops a simulated annealing (SA) algorithm in which three decode methods (list scheduling, permutation scheduling, and first-fit method) are used to obtain the objective function value for the problem. Additionally, a new neighborhood mechanism is combined with the proposed SA for generating neighbor solutions. The proposed SA is tested on two benchmark problems from the literature. The test, made against five algorithms: genetic and memetic algorithm (GA, MA) from Jouglet et al. (2009), another genetic algorithm from Serifoglu & Ulusoy (2004), an ant colony system optimization (ACS) from Ying & Lin (2006), and a particle swarm optimization (PSO) from Tseng & Liao (2008) shows that the proposed SA is an efficient approach in solving hybrid FSSP with multiprocessor tasks, especially for large problems.

2.2.12 J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse, F. Werner; *Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, (2008) 37: 354–370

In textile industries, production facilities are established as multi-stage production flow shop facilities, where a production stage may be made up of parallel machines. This known as a flexible or hybrid flow shop environment. This paper considers the problem of scheduling *n* independent jobs in such an environment. In addition, they also consider the general case in which parallel machines at each stage may be unrelated. Each job is processed in ordered operations on a machine at each stage. Its release date and due date are given. The preemption of jobs is not permitted. They consider both sequence- and machine-dependent setup times. The problem is to determine a schedule that minimizes a convex combination of makespan and the number of tardy jobs. A 0–1 mixed integer program of the problem is formulated. Since this problem is NP-hard in the strong sense, we develop heuristic algorithms to solve it approximately. Firstly, several basic dispatching rules and well-known constructive heuristics for flow shop makespan scheduling problems are generalized to the problem under consideration. They sketch how, from a job sequence, a complete schedule for the flexible flow shop problem with unrelated parallel machines can be constructed. To improve the solutions, polynomial heuristic improvement methods based on shift moves of jobs are applied. Then, genetic algorithms are suggested. Then they discuss the components of these algorithms and test their parameters.

2.2.13 S. Rathinasamy, R. R; *Sequencing and scheduling of nonuniform flow pattern in parallel hybrid flow shop.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, (2010) 49: 213–225

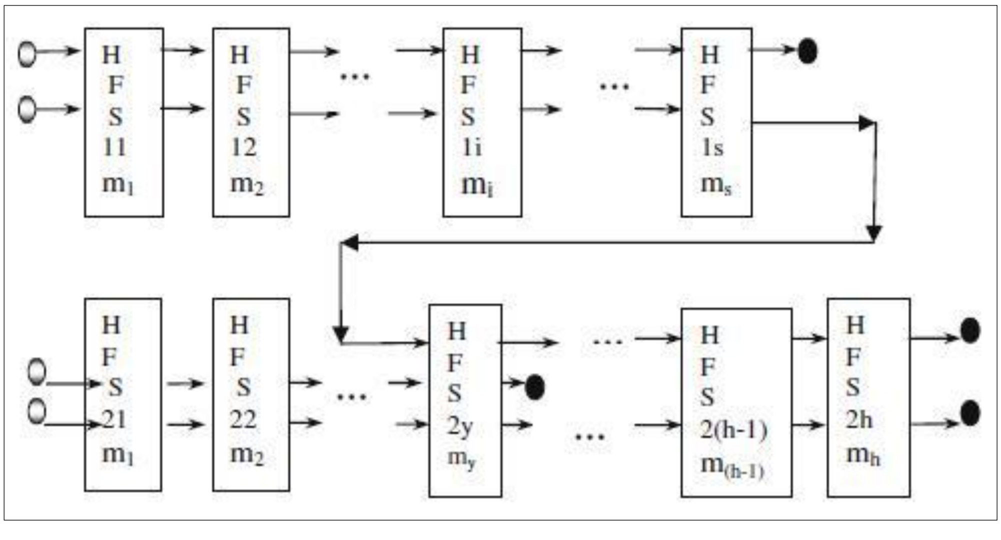


Fig. 2.8: An example of nonuniform flow pattern in parallel HFS, Rathinasamy & R (2010)

This work is motivated by a practical need for operation scheduling for an automobile manufacturing industry involved with machining of different types of vibration dampers. This work aims to minimize the makespan time for relatively different types of vibration dampers with nonuniform flow pattern through a two-parallel inseparable hybrid flow shop. A mathematical model has been developed for simulation of the hybrid flow line performance. The simulation provides data relating to completion time, queue status, and machine utilization of various schedules. After analysis and evaluation of different possible schedules, simultaneous processing of two types of dampers is suggested. The model was evaluated for nonuniform flow pattern for any number of types of dampers for different sets, and a parallel solution based on completion time is suggested. At the end an heuristic is then purposed to provide optimal sequencing for any number of dampers processing in parallel.

2.2.14 M. Haouari, L. Hidri, A. Gharbi; *Optimal scheduling of a two-stage hybrid flow shop.* MATHEMATICAL METHODS OF OPERATIONS RESEARCH, (2006) 64: 107–124

In this paper, it's presented an effective branch-and bound algorithm which has been specifically designed for solving the F2(P) $||C_{max}$ problem with an arbitrary number of machines in each stage; more than two (for each stage) to be accurate. The objective is to schedule a set of jobs so as to minimize the makespan. The authors type that this is the first exact procedure which has been specifically designed for this strongly NP-hard problem. Among other features, the algorithm is based on the exact solution of identical parallel machine scheduling problems with heads and tails. In order to take advantage of the symmetry of the F2(P) $||C_{max}$ problem they do a cyclic implementation, which consists in iteratively solving the *Forward* and the *Backward* problem (the problem, and it's symmetric one), If the branch-and-bound algorithm fails in finding an optimal solution within a given time limit for the *Forward* problem, then it is applied to the *Backward.* The process continues until a solution is proved optimal or there is no improvement of neither the lower nor the upper bound. The results of extensive computational experiments show that the proposed algorithm solves large-scale instances in moderate CPU time.

2.2.15 C. Oguz, M. Erkan; *A genetic algorithm for hybrid flow shop scheduling with multiprocessor tasks.* JOURNAL OF SCHEDULING, (2005) 8: 323–351

The hybrid flow-shop scheduling problem with multiprocessor tasks finds its applications in real-time machine-vision systems among others. Motivated by this application and the computational complexity of the problem, a developed genetic algorithm is proposed. Some assumptions are considered, making the problem really similar to the more diffused paper of the HFS problem: 1. All processors and all jobs are available from time t=0; 2. Processors used at each stage cannot process tasks corresponding to any other stages; 3. Each processor can process not more than one job at a time; 4. Preemption of jobs is not allowed. It is first described the implementation, which include a new crossover operator. This introduced new crossover operator (NXO) for the proposed genetic algorithm, capture some characteristics of $Fk(Pm_1, \ldots , Pm_k)|size_{i\ j}|C_{max}$ problem. The aim in developing NXO is to keep the best characteristics of the parents in terms of the neighboring jobs. Then they perform a preliminary test to set the best values of the control parameters, namely the population size, crossover rate and mutation rate. Next, given these values, they carry out an extensive computational experiment to evaluate the performance of four versions of the proposed genetic algorithm in terms of the percentage deviation of the solution from the lower bound value. The results of the experiments demonstrate that the genetic algorithm performs the best when the new crossover operator is used along with the insertion mutation. All data for the computational experiments were generated randomly in line. This genetic algorithm also outperforms the tabu search algorithm proposed in the literature for the same problem by Oguz et al. (2004).

2.2.16 A. J. Vakharia, J. P. Moily, Y. Huang; *Evaluating virtual cells and multistage flow shops: an analytical approach.* THE INTERNATIONAL JOURNAL OF FLEXIBLE MANUFACTURING SYSTEMS, (2000) 11: 291–314

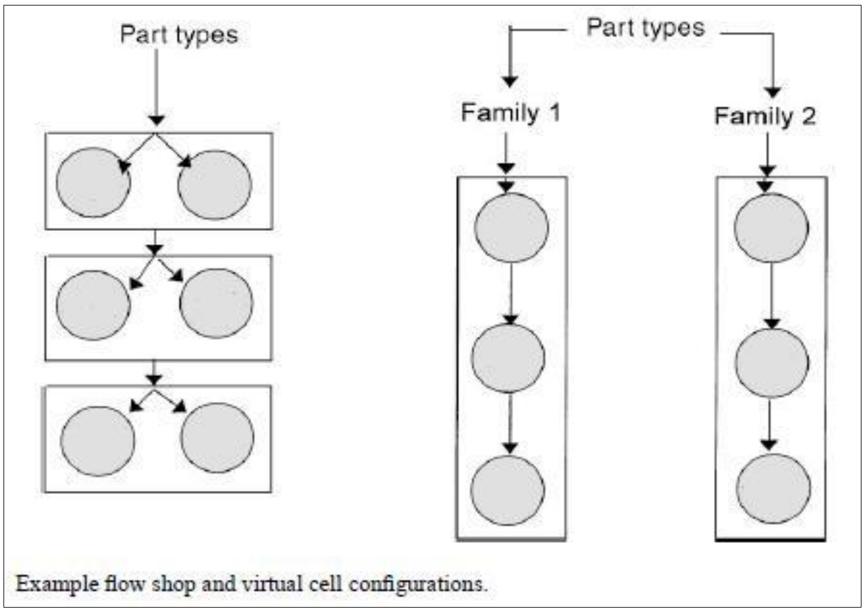


Fig. 2.9: Vakharia et al. (2000)

This is another different solution and perhaps totally different vision of the HFS problem, virtual cells are here compared with the flow shop, that's an important occasion to understand and explain these confronted approaches. The implementation of cellular manufacturing can be carried out through the creation of manufacturing cells (i.e., groups of dissimilar machines dedicated to a set of part types that are placed in close proximity to one another) or virtual cells (i.e., the dedication of specific machines within the current departments to a prespecified set of part types). Typically, the former involves the reorganization of the shop floor and provides the operational benefit of reduced materials handling. On the other hand, the latter configuration is simpler to implement and easier to reconfigure in light of product demand changes, but it may not offer the same operational benefits. In this paper, they propose and validate analytical approximations for comparing the performance of virtual cells and multistage flow shops. Using these approximations and hypothetical data, some key factors that influence the implementation of virtual cells in a multistage flow shop environment are individuated. All it's ended with an application of the presented approximations to industrial data, to show that the efficiency of the virtual cells is a function of how machines at each processing stage are dedicated.

2.2.17 A. R. Rahimi-Vahed, S. M. Mirghorbani; *A multi-objective particle swarm for a flow shop scheduling problem.* JOURNAL OF COMBINATORIAL OPTIMIZATION, (2007) 13: 79–102

Flow shop problems as a typical manufacturing challenge have gained wide attention in academic fields. In this paper, they consider a bi-criteria permutation flow shop scheduling problem, where weighted mean completion time and weighted mean tardiness are to be minimized simultaneously. Since a flow shop scheduling problem has been proved to be NP-hard in strong sense, an effective multi-objective particle swarm (MOPS), exploiting a new concept of the Ideal Point and a new approach to specify the superior particle's position vector in the swarm, is designed and used for finding locally Pareto-optimal frontier of the problem. Tabu Search was used to generate diverse initial solutions. A similar concept for Ideal Point in multi-objective optimization problems (Dynamic Ideal Point) was introduced and used in the initialization phase and in the main algorithm. In the initialization phase, the DIP was approximated using linear programming when finding the exact Ideal Point was a difficult task and this approximation was improved with regard to the better values found for each of the objective functions throughout the main algorithm A new method was applied to specify the superior particle position's vector (*Pg*) in the swarm based on solutions' crowding distance rather than dominance concept. In all test problems, the MOPS was able to improve the quality of the obtained solutions, especially for the large-sized problems and in five cases in the small sized problems, all (100%) of the available non-dominated solutions of the searching space obtained by enumeration was detected by MOPS. To prove the efficiency of the proposed algorithm, various test problems are solved and the reliability of the proposed algorithm, based on some comparison metrics, is compared with a distinguished multi-objective genetic algorithm, the strength Pareto evolutionary algorithm II (SPEA-II) proposed and presented by Zitzler et al. (2001), showing this primary result: MOPS outperforms the compared GA specially in large-sized environments.

2.2.18 H.-S. Choi, D.-H. Lee; *Scheduling algorithms to minimize the number of tardy jobs in two-stage hybrid flow shops.* COMPUTERS & INDUSTRIAL ENGINEERING, (2009) 56: 113–120

A two-stage hybrid flow shop scheduling problem for the objective of minimizing the number of tardy jobs is here presented. Each job is processed through the two production stages in series, each of which has multiple identical parallel machines. The problem is to determine the allocation of jobs to the parallel machines as well as the sequence of the jobs assigned to each machine. To solve the problem, a branch and bound algorithm, which incorporates the methods to obtain the lower and upper bounds as well as the dominance properties to reduce the search space, is suggested to give the optimal solutions. In addition, two-phase heuristic algorithms are suggested to obtain good solutions for large-size problems within a reasonable amount of computation time. In the first phase, the ready time of each job at the first stage is obtained using a backward schedule, and then in the second phase, it is changed to a better schedule if any of the ready times is negative. This paper suggests six heuristic algorithms according to the methods used in the second phase. To show the performances of the optimal and heuristic algorithms suggested in this paper, computational experiments are done on a number of randomly generated test problems, and the test results are reported and compared with the previous works of Gupta (1988) on the heuristics. The authors purpose to develop the models into several directions: more than two stages, uniform or unrelated parallel machines, or reentrant product flow.

2.2.19 M. R. Amin-Naseri, M. A. Beheshti-Nia: *Hybrid flow shop scheduling with parallel batching.* INTERNATIONAL JOURNAL OF PRODUCTION ECONOMICS, (2009) 117:185–196

There are two types of batch productions, namely, serial batches and parallel batches. In serial batches a number of jobs within the same batch are processed sequentially, while in parallel batches a group of jobs go through a machine and are processed simultaneously (Xuan & Tang, 2007). In this research the problem of parallel batch scheduling in a hybrid flow shop environment with minimizing C_{max} is studied . In parallel batching it is assumed that machines in some stages are able to process a number of operations simultaneously. Since the problem is NP-hard, three heuristic algorithms called H1, H2 and H3 are developed to give near optimal solutions. In this section three heuristic algorithms are developed in order to solve the problem. The first heuristic *(H1)* is based on Johnson's rule. The second heuristic *(H2)* is inspired by parallel machine scheduling techniques. The third heuristic *(H3)* is based on the theory of constraints. The solution process in all heuristics is implemented through two phases: (1) sequencing operations and (2) scheduling operations, in which operations are assigned to machines and their start and finish time are computed. Since this problem has not been studied previously, therefore, a lower bound is developed for evaluating the performance of the proposed heuristics. Several test problems have been solved using these heuristics and results compared. To further enhance the solution quality, a three dimensional genetic algorithm (3DGA) is also developed. Several test problems have been solved using 3DGA and the results indicate its superiority against an heuristic algorithm called NEH proposed by Nawaz et al. (1983), and the four heuristics proposed previously. A lower bound (branch and bound) algorithm was also implemented to compare the previous solutions with exact ones. The authors type that a scope for the future researches could be the combination of serial and parallel batching.

2.2.20 S. Khalouli, F. Ghedjati, A. Hamzaoui; *A meta-heuristic approach to solve a JIT scheduling problem in hybrid flow shop*. ENGINEERING APPLICATIONS OF ARTIFICIAL INTELLIGENCE, (2010) 23: 765–771

The increase of competitiveness has motivated the implementation of just in time (JIT) production on scheduling problem to reduce process inventories and delivering goods at time. In fact, this production environment is benefit to both manufacturers and customers. In this paper they address a hybrid flow shop scheduling problem considering the minimization of the sum of the total earliness and tardiness penalties. Their objective is to introduce the problem of extra time (ET) costs into the HFS scheduling problem typical solution. This is proven to be NP-hard, and consequently the development of heuristic and meta-heuristic approaches to solve it is well justified. So, they propose an ant colony optimization *(ACO-HFSET)* method to deal with this problem. The proposed method has several features, including some heuristics that specifically take into account both earliness and tardiness penalties to compute the heuristic information values. The performance of this algorithm is tested by numerical experiments on a large number of randomly generated problems. A comparison with solutions performance obtained by some constructive heuristics (CH) is presented. The results show that the proposed approach performs well for this problem.

2.2.21 M. Zandieh, S. M. T. Fatemi Ghomi, S. M. Moattar Husseini; *An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times.* APPLIED MATHEMATICS AND COMPUTATION (2006) 180: 111–127

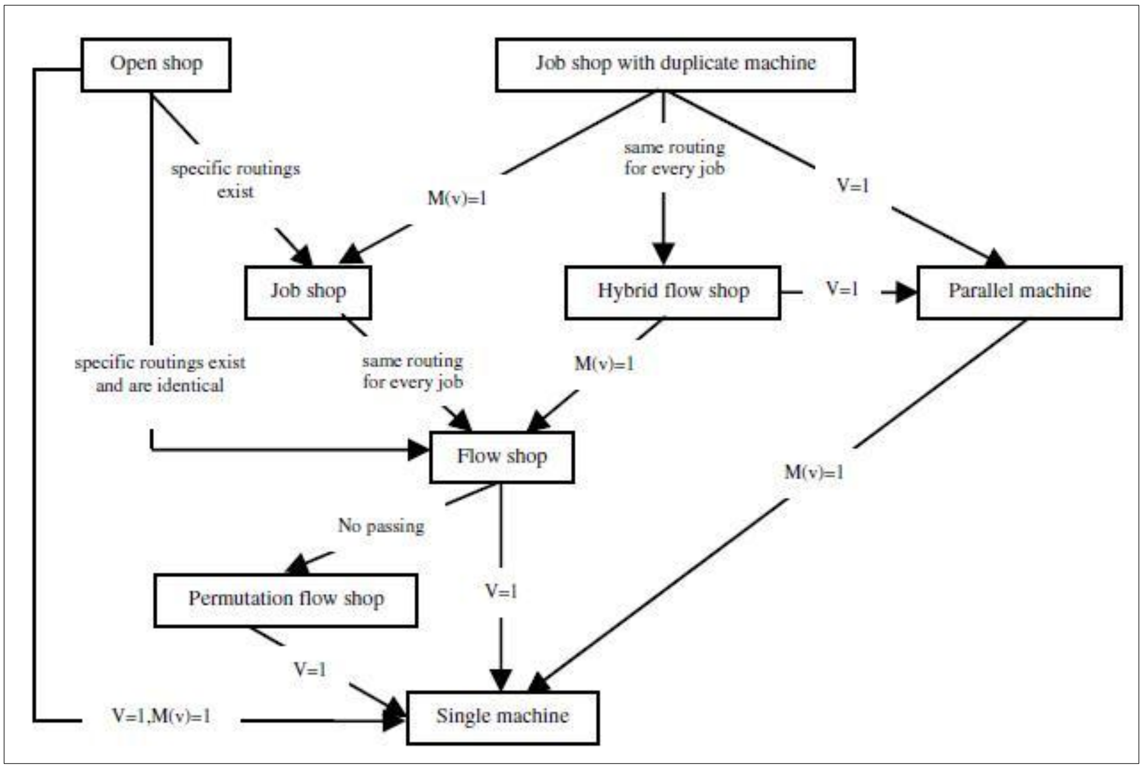


Fig. 2.10: A classification of scheduling problems based on resource environments, Zandieh et al. (2006)

Much of the research on operations scheduling problems has either ignored setup times or assumed that setup times on each machine are independent of the job sequence. This paper deals with the hybrid flow shop scheduling problems in which there are sequence dependent setup times, commonly known as the SDST hybrid flow shops, with the objective of minimizing makespan. This type of production system is found in industries such as chemical, textile, metallurgical, printed circuit board, and automobile manufacture. With the increase in manufacturing complexity, conventional scheduling techniques for generating a reasonable manufacturing schedule have become ineffective. An immune algorithm (IA) can be used to tackle complex problems and produce a reasonable manufacturing schedule within an acceptable time. This paper describes an immune algorithm approach to the scheduling of a SDST hybrid flow shop. An overview of the hybrid flow shops and the basic notions of an IA are first presented. The natural immune system is a very complex system with several mechanisms to defense against pathogenic organisms. However, the natural immune system is also a source of inspiration for solving optimization problems. From the information processing perspective, immune system is a remarkable adaptive system and can provide several important aspects in the field of computation (Dasgupta & Attoh-Okine, 1997). When incorporated with evolutionary algorithms, immune system can improve the search ability during the evolutionary process (Jiao & Wang, 2000). Subsequently, the details of an IA approach are described and implemented. The results obtained are compared with those computed by Random Key Genetic Algorithm (RKGA) presented previously by Kurz & Askin (2003,2004). From the results, it was established that IA outperformed RKGA.

2.2.22 H.-T. Lin, C.-J. Liao; *A case study in a two-stage hybrid flow shop with setup time and dedicated machines.* INTERNATIONAL JOURNAL OF PRODUCTION ECONOMICS (2003) 86: 133–143

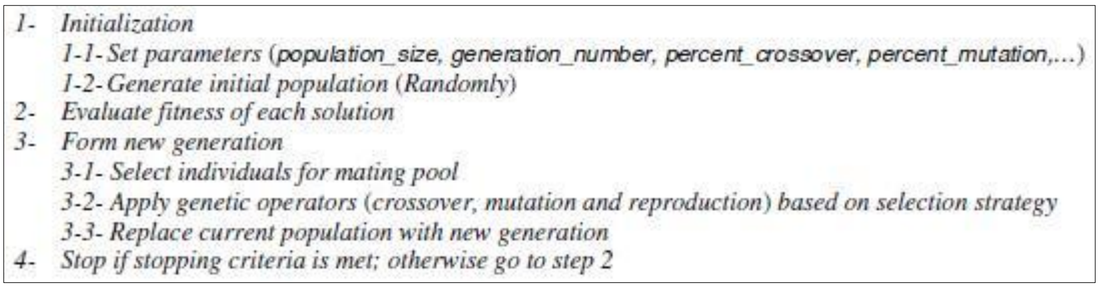
In this paper they address a scheduling problem taken from a label sticker manufacturing company, so a real case. The production system is a two-stage hybrid flow shop with the characteristics of sequence-dependent setup time at stage (1), dedicated machines at stage (2), and two due dates. The objective is to schedule one day’s mix of label stickers through the shop such that the weighted maximal tardiness is minimized. A heuristic is proposed to find the near-optimal schedule for the problem. This heuristic is called modified GD method, because it’s a development of the Gupta and Darrow heuristic algorithm (Gupta & Darrow, 1986). As the heuristic is based on
the specific requirements of the system, it can effectively improve the performance of the system. The performance of the heuristic is evaluated by comparing its solution with both the optimal solution for small-sized problems and the solution obtained by the scheduling method currently used in the shop, a branch and bound (B&B), presented by Linn & Zhang

(1999). The management is currently developing an aggregate production management system, which includes order treatment, scheduling, inventory control, forecasting, and capacity planning modules. As the heuristic is beneficial to the company, it will be used in the scheduling module and implemented in the near future. Although the heuristic is developed for the specific system, it can be used, with appropriate modifications, in other two-stage FSMP scheduling problems with similar features.

2.2.23 S. Voß, A. Witt; *Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: a real-world application.* INTERNATIONAL JOURNAL OF PRODUCTION ECONOMICS (2007) 105: 445–458

Due to intense performance improvements of information technology in the last decade quantitative models have become applicable to reasonably sized real-world scheduling problems. Particularly in supply chain management the partner-companies are in need of extensive tuning of their activities. Apart from a timely data transmission this also presumes extremely current and accurate planning results. The use of quantitative models and procedures lays the foundations for this purpose. To confirm and develop this statement they consider a real world multi-mode multi-project scheduling problem. Furthermore, sequence-dependent setup states arise at two production stages leading to a batching problem. The objective is to minimize the weighted tardiness. Usually, machine scheduling models do not include general precedence constraints between jobs. A promising way to focus on a more general approach is to use the well-known resource constrained project scheduling problem (RCPSP, sometimes referred to as a project shop; Morton & Pentico, 1993) with multiple modes as a basis for the development of an integer model with discrete time periods. The RCPSP is well suited because contrary to traditional models of machine scheduling it A mathematical model based on the well-known RCPSP is presented to provide a formal description of the problem. As problem instances consist of lots of thousands jobs a heuristic solution procedure using dispatching rules is also applied. They describe how these rules can be modified in order to adapt them to a problem with batch requirements. For the RCPSP there exist two different schemes to generate schedules with dispatching rules: so-called serial and parallel generation schemes. First they consider some small test problem instances and then they turn to the real-world instances.

2.2.24 M. Zandieh, E. Mozaffari, M. Gholami; *A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems.* JOURNAL OF INTELLIGENT MANUFACTURING (2010) 21: 731–743



```
1- Initialization
   1-1- Set parameters (population_size, generation_number, percent_crossover, percent_mutation,...)
   1-2- Generate initial population (Randomly)
2- Evaluate fitness of each solution
3- Form new generation
   3-1- Select individuals for mating pool
   3-2- Apply genetic operators (crossover, mutation and reproduction) based on selection strategy
   3-3- Replace current population with new generation
4- Stop if stopping criteria is met; otherwise go to step 2
```

Fig 2.11: A standard genetic algorithm in pseudo-code, Zandieh et al. (2010)

This article addresses the problem of hybrid flexible flow line where some constraints are considered to alleviate the chasm between the real-world industries scheduling and the production scheduling theories. Sequence-dependent setup times (SDST), machine release date and time lags are three constraints deemed to project the circumstances commonly found in real-world industries. A crucial property characteristic of this research topic is that sequence-dependent setup times exist between jobs at each stage. After completing processing of one job and before beginning processing of the next job, some sort of setup must be performed. The length of time required to do the setup depends on both the prior and the current job to be processed; that is, the setup times are sequence-dependent (Kurz & Askin, 2003). To tackle the complexity of the problem at hand, they propose an approach base on genetic algorithm (GA). However, the performance of most evolutionary algorithms is significantly impressed by the values determined for the miscellaneous parameters which these algorithms possess. Hence, response surface methodology is applied to set the parameters of GA and to estimate the proper values of GA parameters in continually intervals. Finally, problems of various sizes are utilized to test the performance of the proposed algorithm and to compare it with some existing heuristic in the literature such as SPT, LPT and NEH (Nawaz et al., 1983).

2.2.25 B. Qian, L. Wang, D.-X. Wang, W.-L. Wang, X. Wang; *An effective hybrid de-based algorithm for multi-objective flow shop scheduling with limited buffers.* COMPUTERS & OPERATIONS RESEARCH (2009) 39: 209–233

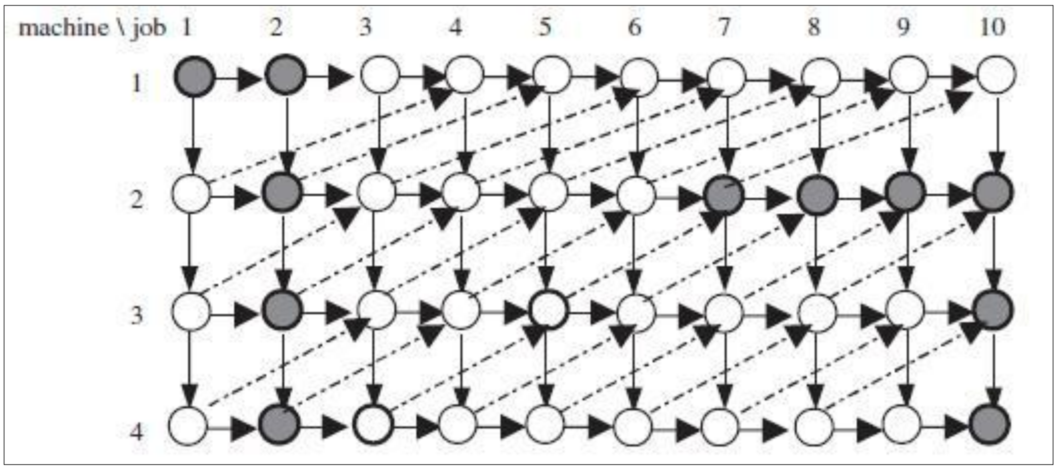


Fig. 2.12: A graph model of flow shop with limited buffers, Nowicki (1999)

This paper proposes an effective hybrid algorithm based on differential evolution (DE), namely HDE, to solve multi-objective permutation flow shop scheduling problem (MPFSSP) with limited buffers between consecutive machines, which is a typical NP-hard combinatorial optimization problem with strong engineering background. Firstly, to make DE suitable for solving scheduling problems, a largest-order-value (LOV) rule is presented to convert the continuous values of individuals in DE to job permutations.
Secondly, after the DE-based exploration, an efficient local search, which is designed based on the landscape of MPFSSP with limited buffers, is applied to emphasize exploitation. Thus, not only does the HDE apply the parallel evolution mechanism of DE to perform effective exploration (global search) in the whole solution space, but it also adopts problem-dependent local search to perform thorough exploitation (local search) in the promising sub-regions. In addition, due to the parallel evolutionary framework of DE, local search is easy to embed in DE to develop effective hybrid algorithms. Next, we will present this local search, which is embedded in DE for solving

MPFSSP with limited buffers. In addition, the concept of Pareto dominance is used to handle the updating of solutions in sense of multi-objective optimization. Moreover, the convergence property of HDE is analyzed by using the theory of finite Markov chain (Pranzo, 2004). Finally, simulations and comparisons based on benchmarks demonstrate the effectiveness and efficiency of the proposed HDE.

2.2.26 C. Kahraman, O. Engin, I. Kaya, R. E. Ozturk; *Multiprocessor task scheduling in multistage hybrid flow-shops: a parallel greedy algorithm approach.* APPLIED SOFT COMPUTING (2010) 10: 1293–1300

Hybrid flow shop scheduling problems have a special structure combining some elements of both the flow shop and the parallel machine scheduling problems. Multiprocessor task scheduling problem can be stated as finding a schedule for a general task graph to execute on a multiprocessor system so that the schedule length can be minimized. Hybrid Flow Shop Scheduling with Multiprocessor Task (HFSMT) problem is known to be NP-hard. In this study is presented an effective parallel greedy algorithm to solve HFSMT problem. Parallel greedy algorithm (PGA) is applied by two phases iteratively, called destruction and construction. In the destruction procedure; *d* jobs are randomly chosen in the *n* jobs sequence, where *d* is showing the number of subgroups. Subgroup number has a very important role in PGA and it helps to maintain diversity in the population. Subgroup number can be generated from the range (1;n−1). In this study subgroup number is tested for HFSMT problems using the range (1;n−1). The initial population size is divided by two and thus two separate subpopulations with equal size are obtained. For parallel calculating, exactly two subpopulations are chosen. The construction and destruction methods are applied to all subpopulations job sequence. In this study, the population size is accepted as the permutation (π) of *n* jobs. Local search algorithm is applied based on the insertion neighborhood, which is commonly regarded as a very good choice for the scheduling problem (Ruiz & Stutzle, 2007) Four constructive heuristic methods are proposed to solve HFSMT problems. A preliminary test is performed to set the best values of control parameters, namely population size, subgroups number, and iteration number. The best values of control parameters and operators are determined by a full factorial experimental design using our PGA program. Computational results are compared with the earlier works of Oguz et al. (2004,2005) and Oguz (2006). The results indicate that the proposed parallel greedy algorithm approach is very effective in terms of reduced total completion time or makespan (C_{max}) for the attempted problems.

2.2.27 C. Koulamas, G. J. Kyparisis; *A note on the two-stage assembly flow shop schedulino problem with uniform parallel machines.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH 182 (2007) 945–951

The problem of minimizing the makespan in a two-stage assembly flow shop scheduling problem with uniform parallel machines is here presented. This research propose a small survey on the previous methods and papers that inspired their research, for example Cheng & Wang (1999) considered a related problem of scheduling the fabrication and assembly of components in a two-machine flowshop so as to minimize the makespan. Each jobs consists of a component unique to that job processed individually on the first machine and a component common to all jobs processed in batches on the first machine with a setup needed to form each batch. The assembly operation of a job is performed on the second machine. Cheng & Wang (1999) show that this problem is NP-complete with either batch availability or item availability for the common components. They also identify several properties of an optimal solution and some polynomially solvable cases, giving some base concepts for this research, that obviously develop all of this. This problem is a generalization of the assembly flow shop problem with concurrent operations in the first stage and a single assembly operation in the second stage. The solution method is an heuristic algorithm with an absolute performance bound which becomes asymptotically optimal as the number of jobs becomes very large. To their knowledge, this is the first paper in the literature which studies the $AFQ||C_{max}$ problem. The study is then compared with the simpler assembly flow shop problem (without uniform machines) and with the two-stage hybrid flow shop problem with uniform machines, which is the part of the paper of our interest. When there is only one concurrent operation at stage 1, the AFQ||C_{max} problem reduces to the two-stage flexible or hybrid flow shop problem with uniform parallel machines, denoted as HF2Q||C_{max}. The H_0 heurtistic algorithm can perform both the AFQ||C_{max} and the HF2Q||C_{max}, theoretical and pratical assumptions are presented to explain all of this more clearly, some of these are based on Sevastianov (2002).

2.2.28 P. Caricato, A. Grieco, D. Serino; *Tsp-based scheduling in a batch-wise hybrid flow-shop.* ROBOTICS AND COMPUTER-INTEGRATED MANUFACTURING (2007) 23: 234–241

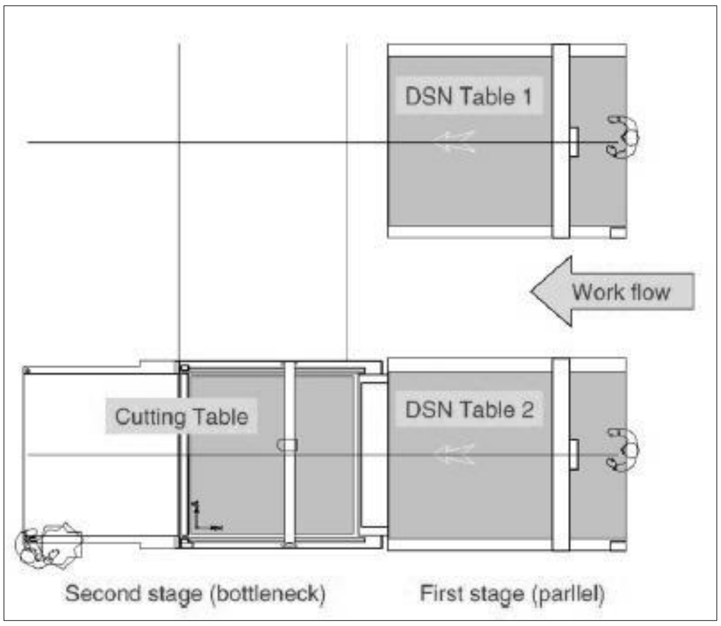


Fig 2.13: System Layout, Caricato et al. (2005)

The problem presented in this paper extends the HFS scheduling problem to cover the case in which the system's behavior is hardly influenced by a batch organization of jobs. In the past few years, flexible production systems have allowed an extensive exploitation of new technologies, but have also led to new difficulties in production planning management science. The model presented in this paper extends the traditional HFS (hybrid flowshop) scheduling problem to the case in which jobs are due to follow strict precedence constraints and batch assignment constraints and the parallel machines at a stage are served by a bottleneck machine. A variant of the well-known travelling salesman problem (TSP) is used to develop an efficient heuristic solution for the problem. The effectiveness of the proposed approach is validated through a comparison with different heuristics traditionally used in HFS scheduling problems, such as: rule based scheduling heuristic, TSP-based simple insert heuristic and TSP-based GENIUS procedure from Mladenovic & Hansen (1997). A thorough analysis of the TSP can be found in Gutin & Punnen (2002). Furthermore, a simple insertion heuristic based on the TSP model of the problem is tested. Finally, a MIP-based approach is also explored to provide the optimum solutions within much larger times for comparison, for more informations about this last presented approach see Caricato et al. (2001).

2.2.29 H.-S. Choi, J.-S. Kim, D.-H. Lee; *Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a TFT-LCD line.* EXPERT SYSTEMS WITH APPLICATIONS (2010) (Article in Press)

Hybrid flow shops can be classified into two types according to product flows: (a) those with unidirectional flows; and (b) those with reentrant flows. Here, the unidirectional flows imply that each job starts at the first stage and finishes at the last stage. On the other hand, in the reentrant flows, each job may visit each serial stage two or more times. For example, semiconductor wafer fabrication and TFT-LCD manufacturing lines have the reentrant flows. In other words, each visit of certain specified serial production stage corresponds to a layer that is built up for required circuits.
Compared with the unidirectional flows, the reentrant flows generally make system operations much more complicated. This paper focuses on the scheduling problem in hybrid flow shops with reentrant product flows, called *reentrant hybrid flow shop scheduling* in this paper. The main decisions are: (a) allocation of jobs to machines at each stage; and (b) sequence of the jobs assigned to each machine. In fact, this research was motivated from a TFT-LCD manufacturing system with a large number of complex processes and reentrant product flows. Unlike the theoretical approach on reentrant hybrid flow shop scheduling, a realtime scheduling mechanism with a decision tree when selecting appropriate dispatching rules is suggested. The

decision tree, one of the commonly used data mining techniques, is adopted to eliminate the computational burden required to carry out simulation runs to select dispatching rules. To illustrate the mechanism suggested in this study, a case study was performed on a thin film transistor-liquid crystal display (TFT-LCD) manufacturing line and the results are reported for various system performance measures.

2.2.30 T. Sawik; *An Exact Approach for Batch Scheduling in Flexible Flow Lines with Limited Intermediate Buffers.* MATHEMATICAL AND COMPUTER MODELLING (2002) 36: 461-471

The paper presents a mixed integer programming approach for makespan mmimization in flexible flow lines. The line consists of several processing stages in series, separated by finite intermediate buffers, where each stage has one or more parallel identical machines. The problem objective is to determine a minimum length schedule for a mix of part types, where identical parts are scheduled consecutively. A mixed integer programming model is presented for batch scheduling in a flexible flow line with limited intermediate buffers. A unified modeling approach is adopted with the buffers viewed as machines with zero processing times. As a result, the scheduling problem with buffers can be converted into one with no buffers but with blocking (Sawik 2000, McCormick et al. 1989). The blocking time of a machine with zero processing time denotes part waiting time in the buffer represented by that machine. They assume that each part must be processed in all stages, including the buffer stages. However, zero blocking time in a buffer stage indicates that the corresponding part does not need to wait in the buffer. Let us note that for each buffer stage part completion time is equal to its departure time from the previous stage since the processing time is zero. Numerical examples modeled after real-world surface mount technology lines for printed wiring board assembly are provided and some computational results are reported to illustrate the approach. The presented approach (based on a MIP formulation) is capable of optimal scheduling of batches of different part types by using commercially available software for discrete programming. The mathematical formulation includes various cutting constraints exploiting special FFL configurations and some properties of batch processing on parallel machines. The cutting constraints have an impact on reducing computational effort required to find the optimal solution. Nevertheless, the CPU time required to find proven optimal schedules for realistic large size problems still can be very high.

2.3 Survey and discussion of the papers

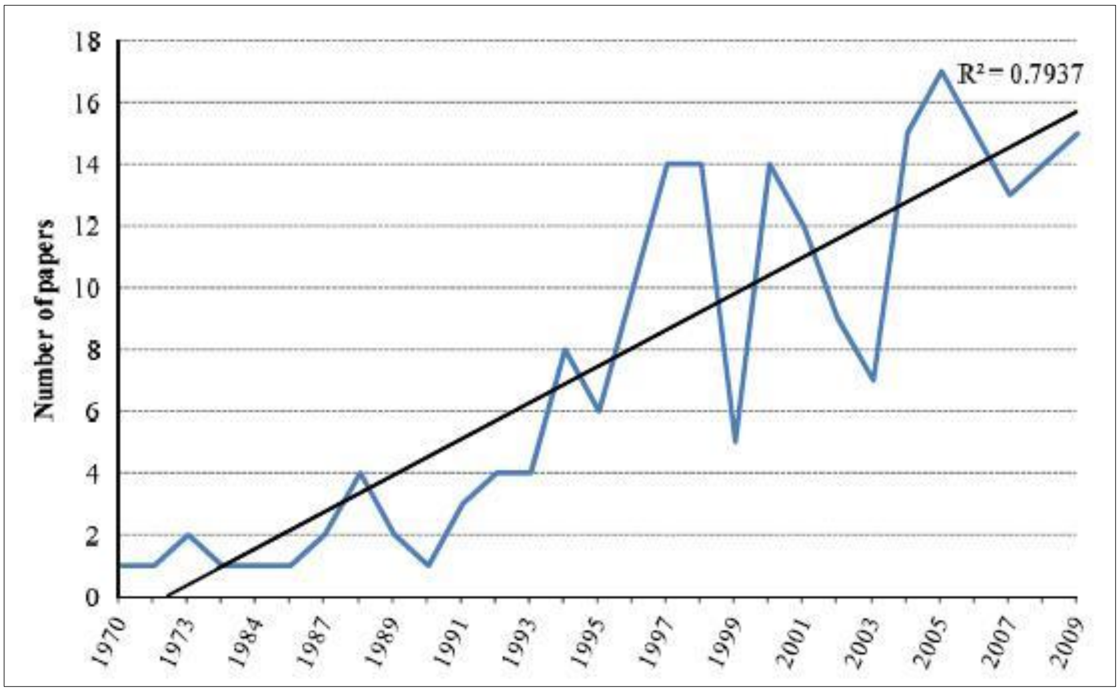
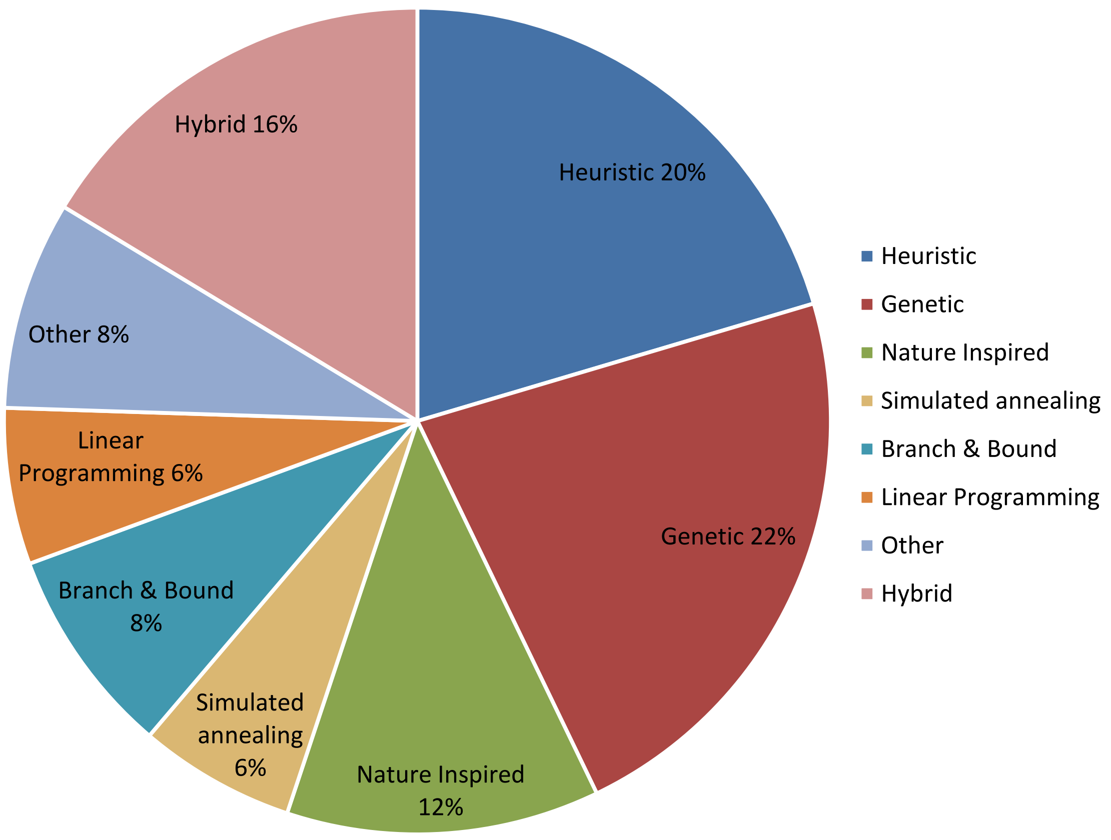


Fig. 2.14: Evolution of number of papers per year; Ruiz & Vazquez-Rodriguez (2010).

First of all, is very important to underline that a research of thirty papers can't give a complete survey, neither give numbers or percentage, etc.. But graphics and tables are very clear instruments to show what have been done, and what can be done to improve this work, so it have been decided to use a short survey to end the second part of the thesis, we will deepen into the simpler objects of research: algorithms and optimized parameter/s; using sometimes the same scheme of two important and recent surveys found in the literature: *Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective*, from Imma Ribas, Rainer Leinsten, Jose M. Framinan (see

Ribas et al., 2009) and *The hybrid flow shop scheduling problem* from Ruben Ruiz and Jose Antonio Vazquez-Rodriguez (see Ruiz & Vazquez-Rodriguez, 2010). Another important paper could be *Hybrid Flow Shop Scheduling: a survey*, from Richard Linn and Wei Zhang (see Linn & Zhang, 1999) but being wrote into 1999, the purposed procedure was not adaptive to the recent papers in this research, but despite that, looking at the papers bibliography this article is so considered from the authors and it's correct to mention it. To end this introduction we invite to focus on fig. 2.14, this graphic, show easily how important is this question for the literature, and that the number of papers grows year by year.

2.3.1 Deepen about algorithms



Graph. 1: Distribution of employed techniques

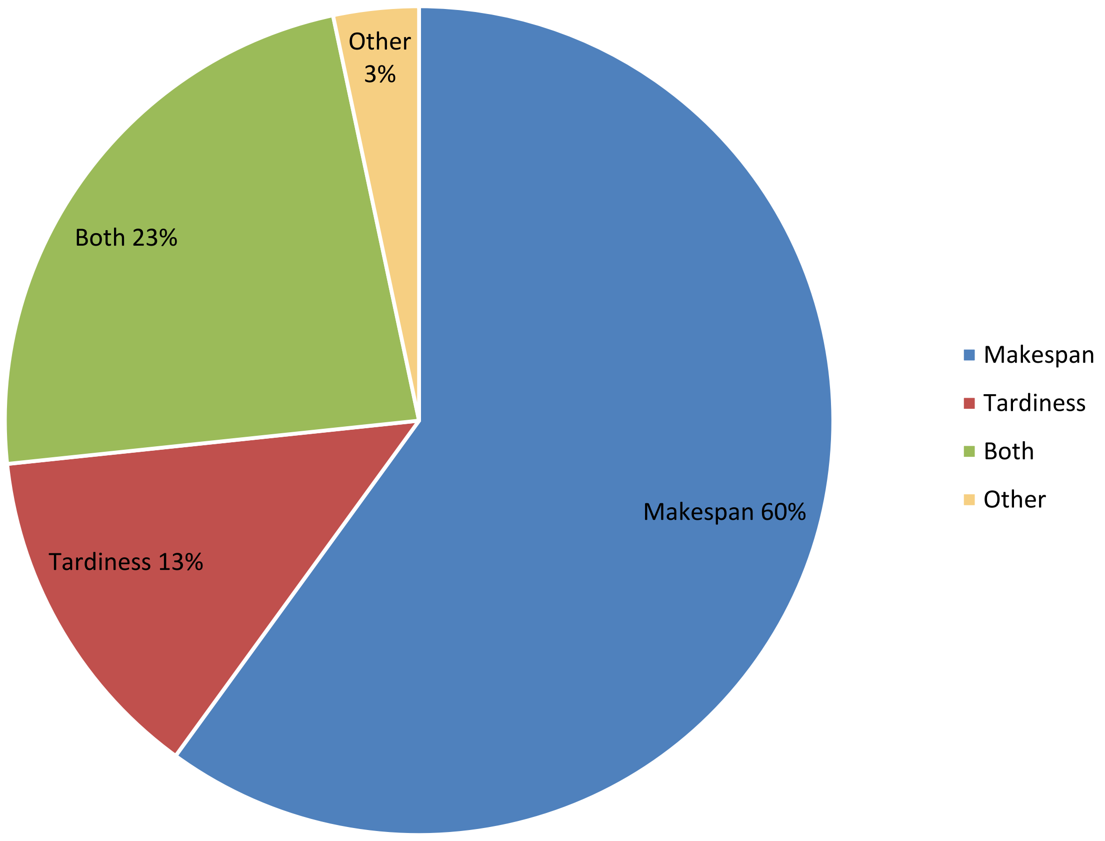
This graphic shows a first division of the algorithms but a lot of other information should be given to estimate the state of art we can see with these thirty papers, despite that, this graphic give fairly good results, if compared with the one from Ruiz & Vazquez-Rodriguez, the percentage of algorithms in a complete survey is similar to the one we purpose, note that this could be just a coincidence, because if the small number of articles. Further information is given from the table 2.2, these table gives the name of the algorithm paper by paper, after the table some notes about algorithms are also given.

Nr.	Algorithm	Nr.	Algorithm
1	Novel simulated annealing (SA)	16	No algorithm (Analytic)
2	Bi-objective heuristic (HE)	17	Multi Object Particle Swarm (NI) (H)
3	Memetic algorithm (GA) (H)	18	Branch and Bound (BB)
4	*No Name* (GA) (H)	19	H1,H2,H3 (HE)
5	Neural Network (NI)	20	Ant Colony Optimization (NI)
6	Random Key Genetic Algorithm (GA)	21	Immune Algorithm (NI)
7	Improved Ant Sysytem (NI)	22	Approximate Algorithm 1 (HE)
8	IHMOPGA (GA) (H)	23	R&M (HE)
9	Hypotesis-Test-Based Genetic Algorithm (GA)	24	Genetic Algorithm (GA)
10	Hybrid Simulated Annealing Algorithm (SA) (H)	25	Hybrid Differential Evolutionary (GA) (H)
11	*No name* (SA)	26	Parallel Greedy (GA)
12	*More than one algorithm*	27	Asymptotically optimal heuristic (H)
13	*No name* (HE)	28	*No name* (GA)
14	Branch and Bound / Lower Bound (BB) (H)	29	Iterative dichotomoser (HE)
15	*No name* (GA) (H)	30	*No name* (LP)

Table 2.2: Name and category of algorithms presented into papers

Note that this last table is not the one used to calculate the graphic, which is the 2.1 Table presented at the beginning of chapter 2.
When you need to put something into a category it isn’t a simple thing to do, some parts could bring you to category “A” some other parts to category “B”, etc.; that’s why a table with all the names of presented algorithms should be given. Note that the Nature Inspired category was invented by me to represent the all algorithms that are inspired from nature, such as ant colony system, particle swarm, neural network etc. I think this can be an interesting name but into the current literature only a few papers use a similar name and it’s correct to say that this can’t be considered a proved and scientific classification.

2.3.2 Objective functions



Graph. 2: Distribution of objective funcions

Notation	Description	Meaning
C_{max}	$\max_j (c_j)$	Maximum completion time
F_{max}	$\max_j (c_j - r_j)$	Maximum flow time
L_{max}	$\max_j (L_j)$	Maximum lateness
T_{max}	$\max_j (T_j)$	Maximum tardiness
E_{max}	$\max_j (E_j)$	Maximum earliness
$\bar{C}^{(w)}$	$\Sigma (w_j c_j)$	Total/average (weighted) completion time
$\bar{F}^{(w)}$	$\Sigma (w_j F_j)$	Total/average (weighted) flow time
$\bar{U}^{(w)}$	$\Sigma (w_j U_j)$	Total/average (weighted) number of late jobs
$\bar{T}^{(w)}$	$\Sigma (w_j T_j)$	Total/average (weighted) tardiness
$\bar{E}^{(w)}$	$\Sigma (w_j E_j)$	Total/average (weighted) earliness

Table 2.3: Common objective functions

The proposed graphic shows a very simple distribution: tardiness and makespan have a lot of different parameters. Common objective functions are presented in table 2.3, note that if the weight is 1 it’s a simple average and not a weighted average, that's why there are parenthesis. At the end we present table 2.4 with the objective functions used into the studied papers. Note that the parenthesis means that the objective function was not as important as the other one/s and that number of tardy jobs have been considered a tardiness parameter into the graphic and into the 2.1 table.

Nr.	C_{max}	T_{max}	$\bar{C}^{(w)}$	$\bar{F}^{(w)}$	$\bar{U}^{(w)}$	$\bar{T}^{(w)}$	$\bar{E}^{(w)}$	Profit
1	X							
2	X					X		
3	X							
4								X
5			X	X	X			
6	X		X					
7	X							
8	X	X						
9	X							
10	X	X						
11	X							
12	X				X			
13	X							
14	X							
15	X							
16			X	X				
17			X			X		
18					X			
19	X							
20						X	X	
21	X		X					
22						X		
23						X		
24	X							
25	X	(X)						
26	X							
27	X							
28	X							
29			(X)	X	X	X		
30	X							

Table 2.4: Objective functions presented into papers

Chapter 3:

Elaboration of two papers of interest

3.1 H.-S. Choi, J.-S. Kim, D.-H. Lee; *Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a TFT-LCD line.* EXPERT SYSTEMS WITH APPLICATIONS (2010) (Article in Press)

3.1.1 Introduction

Most of the previous studies on reentrant hybrid flow shop scheduling are theoretic in the sense that sophisticated algorithms were devised after developing and analyzing mathematical models with various assumptions (See Linn and Zhang (1999) for a literature review on hybrid flow shop scheduling). problem. Real-time scheduling, one of practical scheduling approaches, is an important topic on which a number of previous researches have been done. Unlike the existing approaches explained above, they suggest a real-time scheduling mechanism in which the decision tree is used to select an appropriate dispatching rule at the end of each monitoring period so that the computational burden required for carrying out simulation runs can be eliminated. Here, the monitoring period is the time period during which a dispatching rule is maintained before considering the rule change. Also, the decision tree, a schematic model to determine one of the alternatives available to a decision maker, is constructed using the information obtained from preliminary data. The real-time scheduling mechanism suggested in this paper is illustrated with a case study on a TFT-LCD manufacturing line, and the test results are reported for various system performance measures. Although there have been a number of previous research articles on scheduling in semiconductor manufacturing systems, typical reentrant hybrid flow shops, they have limited applications since they are off-line in nature. Also, the existing real-time scheduling approaches for semiconductor manufacturing select priority dispatching rules using the information obtained from simulation runs and hence they may require significant amount of computational burden. Unlike these, they suggest a real-time scheduling mechanism that increases the speed of the scheduling decisions using the decision tree. Note that the system performances are directly affected by the speed of scheduling system and hence scheduling decisions and actions also have to be made in real-time. This paper is organized as follows: in the second part, they explain the decision tree based real-time scheduling mechanism. The algorithm to construct the decision tree is also explained. The case study on the TFT-LCD manufacturing line is reported in the third section. Finally, the fourth section summarizes the main results, gives the conclusions, and describes some areas for further research. We will add before the second part of the analyzed paper another introductive part, explaining the terms and the methods used in the paper, that are normally considered to be easily known, in the papers, but to an inexpert reader could be difficult to understand.

3.1.2 Real-time scheduling: an historical perspective

A real-time system is one with explicit deterministic or probabilistic timing requirements. Historically, real-time systems were scheduled by cyclic executives, constructed in a rather *ad hoc* manner. During 1970s and 1980s, there was a growing realization that this static approach to scheduling produced systems that were inflexible and difficult to maintain.
Also in this period (starting in 1979), a series of meetings started that eventually turned in to a major annual international conference. The IEEE Real-Time Systems Symposium has in the last twenty-five years been the main forum for publishing the key results in scheduling theory. In the early 1990s a number of other initiatives were funded, including a series of influencial studies commissioned by the European Space Agency. Text book also began to appear, for example, Burns & Wellings (1990). Later, with hardware and software evolution, there has been an explosion of interest in real-time systems, and an explosion of research and pubblication on the analysis of real time scheduling. There are five main categories of real-time scheduling: (1) fixed-priority scheduling, (2) dynamic-priority scheduling, (3) soft real-time scheduling, (4) feedback scheduling and (5) extended scheduling models. (See Sha et al. 2004)

3.1.3 Decision tree based real-time scheduling mechanism

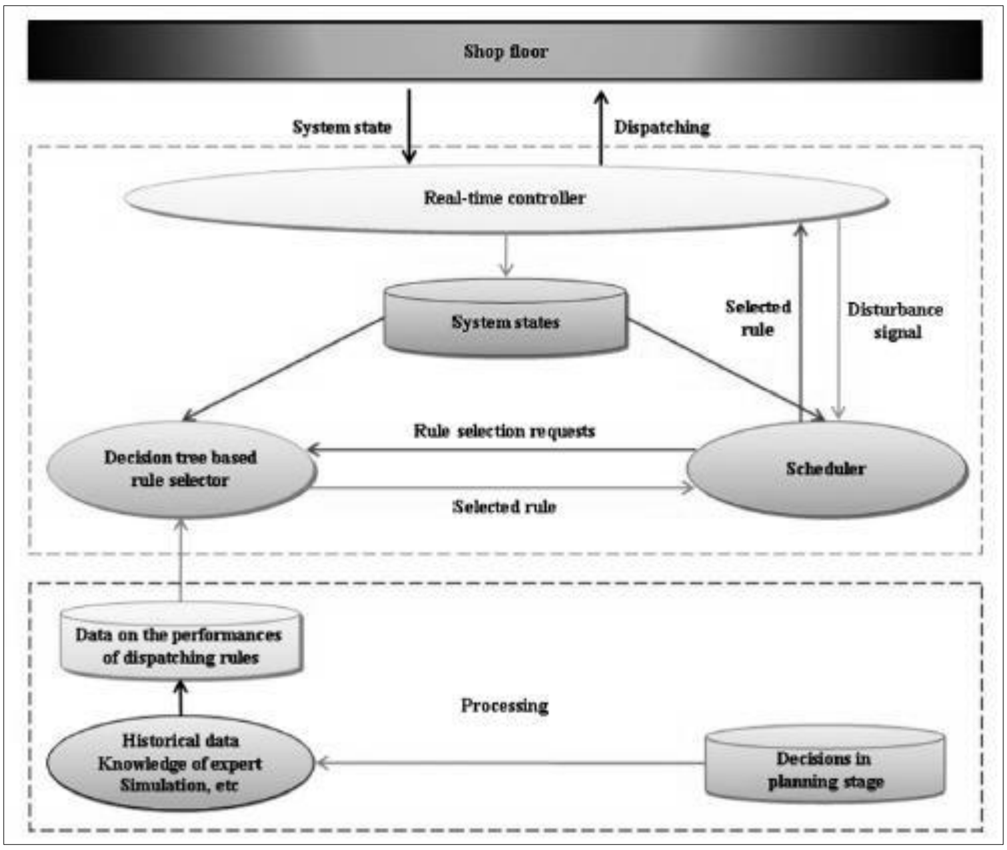

Shop floor
System state
Dispatching
Real-time controller
System states
Selected rule
Disturbance signal
Rule selection requests
Decision tree based rule selector
Scheduler
Selected rule
Data on the performances of dispatching rules
Processing
Historical data Knowledge of expert Simulation, etc
Decisions in planning stage

Fig. 3.1: An overview of the decision tree based real-time scheduling, Choi et al. (2010)

This section presents the decision tree based real-time scheduling mechanism suggested in this paper. First, the framework is explained.
Then, the components and the algorithm to construct the decision tree are explained in details. Finally, the scheduling strategy, i.e., the time point to select a new dispatching rule, is explained.
Fig. 3.1 shows the framework, i.e., components and necessary information for the real-time scheduling mechanism to work. In fact, the framework is a modified version of the simulation-based one of Jeong & Kim (1998) in that the decision tree, instead of simulation, is used to select a new dispatching rule at the end of each monitoring period.
As can be seen in the figure, the real-time scheduling mechanism suggested in this paper consists of three main components: real-time controller, scheduler, and decision tree based rule selector.
A brief explanation of each component is given below. (Details of the components will be explained in the next section.)

- *Real-time controller* exchanges the information with the shop floor, monitors the system states, and dispatches jobs according to the rule released by the scheduler. Also, it updates the system states database using the system monitoring results and sends a signal to the scheduler if it senses the occurrence of a system disturbance.
- *Scheduler* determines the point of time when a new dispatching rule is to be selected. If the real-time controller senses the occurrences of system disturbances and/or a significant difference between real and estimated performances, it sends a signal to the scheduler. This makes a decision on whether or not a new dispatching rule should be selected. When it is necessary to select a new dispatching rule, the scheduler sends a request to the decision tree based rule selector.
- *Decision tree based rule selector* : when the system requests a new dispatching rule, the decision tree based rule selector selects the best dispatching rule, i.e., a rule that gives the best performances, and it is informed to the scheduler.

To explain the relations among the three components, we explain three databases required for our real-time scheduling mechanism to work.

- Decisions in planning stage contain the information about jobs (with operations), routings, processing times, due dates, performance measures, etc.
- System states, updated whenever there is any change in system states, contain the information related to the current system states, i.e., number of jobs in the system, number of remaining operations for each job, processing states of each job, machine states (working, being repaired or idle), etc.
- Data on the performances of dispatching rules contain the information required to build up a decision tree, i.e., system performances under certain system states.

3.1.4 Constructing the decision tree

The decision tree consists of three types of nodes: non-leaf nodes and leaf nodes. Here, each non-leaf node represents a choice among alternatives while leaf nodes represent classification or decision. Before explaining the decision tree in details, an example of the data set is shown in the table, which is adopted from Han (2008).

Objects	*Conditional attributes*				*Decision attribute*
	A	***B***	***C***	***D***	***X***
1	1	2	2	1	1
2	1	2	3	2	1
3	1	2	2	3	1
4	2	2	2	1	1
5	2	3	2	2	2
6	1	3	2	1	1
7	1	2	3	1	2
8	2	3	1	2	1
9	1	2	2	2	1
10	1	1	3	2	1
11	2	1	2	2	2
12	1	1	2	3	1

Table 3.1: Dataset for constructing a decision tree: example, Choi et al. (2010)

In Table 3.1, there are twelve objects, four conditional attributes, and one decision attribute. For example, object 1 implies that the decision is 1 (X = 1) if the values of conditional attribute A, B, C, and D are 1, 2, 2, and 1, respectively. Using the data given in the table, various decision trees can be made. Among them, an example is shown in Fig. 3.2.

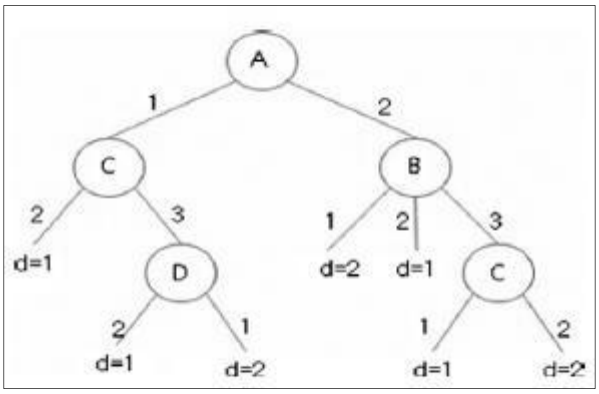


Fig. 3.2 Decision tree: example. Choi et al. (2010)

In Fig. 3.2, a path from the root node to each lead node corresponds to a decision. For example, if the values of conditional attributes A, B and C are 2, 3, and 1, the resulting decision is 1, i.e., X = 1. Now, how the decision tree is used to select a dispatching rule at the end of each monitoring period it's explained. In this application, conditional and decision attributes in the data set correspond to the system states and the selection of dispatching rule, respectively. If simulation is used to construct the decision tree, an object in the data set, i.e., each row in Table 3.1, is obtained by performing a simulation run under a given set of system states and identifying the best dispatching rule. As stated earlier, the data set can be also obtained from the historical data or the knowledge of experts. The system states considered in this study are summarized below. (Note that more variables can be added for other applications.)

- Total number of remaining operations for the jobs in queue at each stage.
- Total processing time of remaining operations for the jobs in queue at each stage.
- Total number of remaining operations for the jobs being processed at each stage.
- Total processing time of remaining operations for the jobs being processed at each stage.

Note that the decision tree can be updated if there are changes in the cumulated data set. This shows the flexibility of our decision tree based real-time scheduling mechanism. There are various algorithms to construct the decision tree. Among them, we adopt the Iterative Dichotomiser algorithm of Quinlan (1986), called the ID3 algorithm in the literature, since it has been proved to be simple but effective to express information contained implicitly in discrete valued data sets.

3.1.5 The iterative dichotomiser (ID3) algorithm

The basic idea of the ID3 algorithm by Quinlan (1986) is stemmed from the information theory and the pattern recognition. Before explaining the algorithm, a set of objects is defined as a matrix A = $[a_{ij}]$, where a_{ij} denotes the value of conditional attribute j of object i. Note that in the matrix, each row vector corresponds to an object without the decision attribute. (See Table 3.1 for an example.)

The ID3 algorithm uses the entropy function to select the conditional attributes of a decision tree, where the entropy function measures the impurity of an arbitrary collection of objects. More formally, the entropy function of conditional attribute j is defined as

- $entropy_j = \sum_{c=1}^{c_j} -p(w_{cj}|j) \cdot \log_2 p\,(w_{cj}|j)$

where C_j denotes the number of different conditional attribute values, e.g., C_A, C_B, C_C, and C_D are, 2, 2, 3, and 3 for the example in the table. Also, $p(w_{cj}|j)$ denotes the proportion of value w_{cj} in conditional attribute j, i.e.,

- $p(w_{cj}|j) = |W_{cj}|/m$

where $W_{cj} = \{i|a_{ij} = w_{cj}, \forall i\}$ and m denotes the number of objects. For example, p(1|A) = 8/12 and p(2|B) = 4/12 in Table 3.1, and hence the entropy value of conditional attribute A can be calculated as follows:

- $entropy_j = -\,{}^{8}/_{12}\,\log_2 {}^{8}/_{12} - {}^{4}/_{12}\,\log_2 {}^{4}/_{12}$

The ID3 algorithm constructs the decision tree as follows. First, all the conditional attributes are evaluated using the entropy function and the one with the smallest entropy value is selected. From the root node, a partial decision tree is constructed with the selected conditional attribute. Second, a child node is generated for each conditional attribute value of the root node and it is connected to the root node. As in the root node, the conditional attribute of the child node is set to the one with the smallest entropy value after removing the selected conditional attribute and the objects with the conditional attribute value of the root node. This is done until there is no remaining conditional attributes to be considered.

A detailed procedure of the ID3 algorithm is given below.

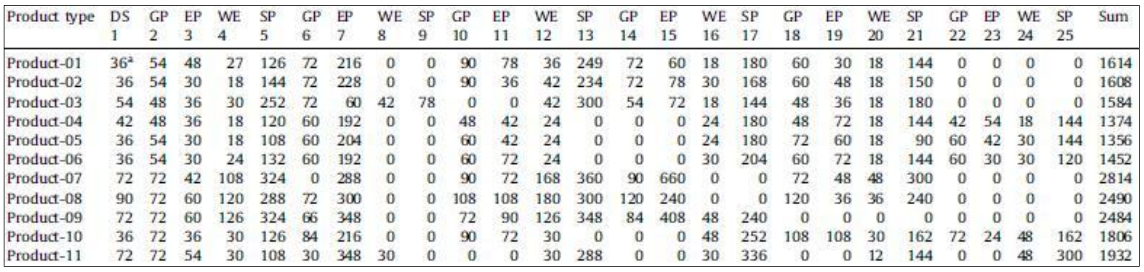
- *Step 1*. Create the root node using the conditional attribute with the smallest entropy value and let the root node be the current node.
- *Step 2*. For each conditional attribute value of the current node, create and connect a child node whose conditional attribute is set to the one with the smallest entropy value after updating the data set, i.e., entropy values are calculated after removing the conditional attribute of the current node and the objects with the conditional attribute value of the current node.

- *Step 3*. If all conditional attributes are considered, stop. Otherwise, let one of the unconsidered child nodes be the current node and go to Step 2.

Afterthe decision tree is constructed, one more decision should be made on the time points when a new dispatching rule is to be selected, i.e., the time when the decision tree is called. To do this, they use the ALL strategy suggested by Jeong and Kim (1998) since it is better than the others. (See Jeong and Kim (1998) for the other scheduling strategies.) In the ALL strategy, the scheduler is called in the following cases.

- Beginning of a new scheduling horizon.
- Major system disturbances (e.g., machine breakdowns).
- Minor system disturbances (e.g., tool breakages).
- Getting the performances worse, i.e., certain performance value exceeds a pre-determined limit, at each periodic monitoring period.

3.1.6 Application on a TFT-LCD line



Product type	DS	GP	EP	WE	SP	GP	EP	WE	SP	GP	EP	WE	SP	GP	EP	WE	SP	GP	EP	WE	SP	GP	EP	WE	SP	Sum
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	
Product-01	36[a]	54	48	27	126	72	216	0	0	90	78	36	249	72	60	18	180	60	30	18	144	0	0	0	0	1614
Product-02	36	54	30	18	144	72	228	0	0	90	36	42	234	72	78	30	168	60	48	18	150	0	0	0	0	1608
Product-03	54	48	36	30	252	72	60	42	78	0	0	42	300	54	72	18	144	48	36	18	180	0	0	0	0	1584
Product-04	42	48	36	18	120	60	192	0	0	48	42	24	0	0	0	24	180	48	72	18	144	42	54	18	144	1374
Product-05	36	54	30	18	108	60	204	0	0	60	42	24	0	0	0	24	180	72	60	18	90	60	42	30	144	1356
Product-06	36	54	30	24	132	60	192	0	0	60	72	24	0	0	0	30	204	60	72	18	144	60	30	30	120	1452
Product-07	72	72	42	108	324	0	288	0	0	90	72	168	360	90	660	0	0	72	48	48	300	0	0	0	0	2814
Product-08	90	72	60	120	288	72	300	0	0	108	108	180	300	120	240	0	0	120	36	36	240	0	0	0	0	2490
Product-09	72	72	60	126	324	66	348	0	0	72	90	126	348	84	408	48	240	0	0	0	0	0	0	0	0	2484
Product-10	36	72	36	30	126	84	216	0	0	90	72	30	0	0	0	48	252	108	108	30	162	72	24	48	162	1806
Product-11	72	72	54	30	108	30	348	30	0	0	0	30	288	0	0	30	336	0	0	12	144	0	0	48	300	1932

Table 3.2: Product routes and processing times (in minutes) for the case studied, Choi et al. (2010)

TFT-LCDs are high-tech display products manufactured through complex processes. A glass of semiconductor material is coated with a thin film of a chemical called photo-resist. Photo-resist coated wafers or glasses are then baked in an oven to remove solvents. Once the baking process is completed, the stepper aligns layers with mask plate and the glass is exposed to ultraviolet light. Then, the glass is developed in the developer. At the final stage, dry and wet etching processes remove thin film layers. The dry etching process uses reactive species, such as atoms or radicals from the gas plasma, to etch away a portion of the object material. When these species react with the material located on the plate, the open region of material is transformed into a volatile state and removed from the matrix. In this process, the reaction velocity is fast, and fine patterns can be formed uniformly. The TFT-LCD fabrication process, a typical bottleneck among the whole processes, is similar to the semiconductor wafer fabrication process in that its complexity comes from a large number of operations as well as reentrant flows. The TFT-LCD fabrication process considered in this study can be described as Fig. 3.3.

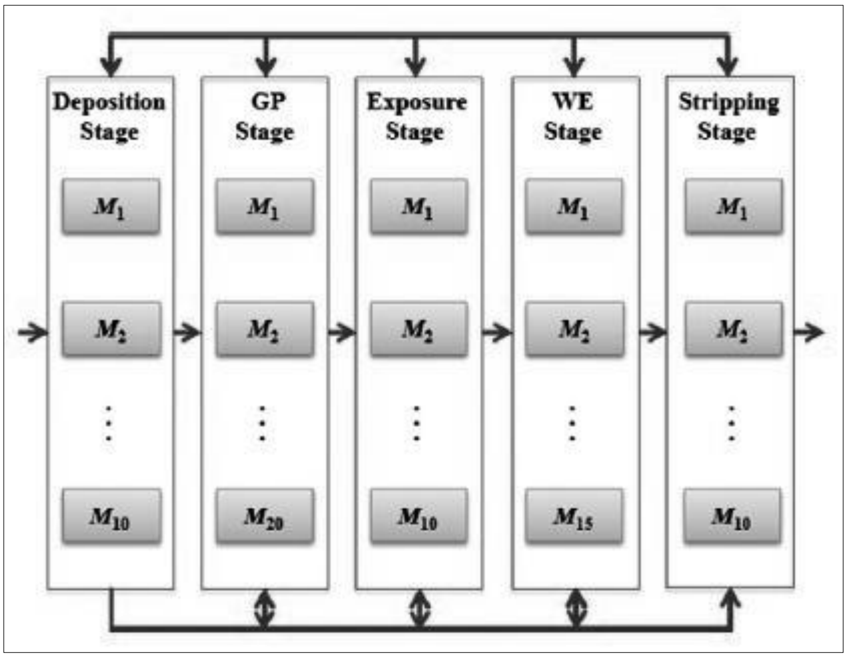


Fig 3.3: TFT-LCD manufacturing process, Choi et al. (2010)

As can be seen in the figure, there are five serial stages, called deposition (DS) with 10 machines, gate photo (GP) with 20 machines, exposure (EP) with 10 machines, wet etching (WE) with 15 machines, and stripping (SP) with 10 machines, in the line. In the TFT-LCD manufacturing line, 11 product types are produced. The routes and processing times are summarized in Table 3.2 According to the operations managers of the line, due dates of jobs were generated from DU(2.0 · Ti, 4.0 · Ti), where DU(l, u) and T denote a discrete uniform distribution with range [l, u] and the sum of the operation times of the job i, respectively. Preemption is not allowed due to the technical problems. It is assumed that the transportation time is ignored since the material handling system is not the bottleneck in the line, and set-up times are included in the processing times. Finally, the other problem data are summarized below. Note that some of the data are artificial due to the confidential problem and the difficulties to obtain the exact data.

- Jobs arrive with an inter-arrival time generated from EXP(10), where EXP(λ) is an exponential distribution with a mean of k.
- Major machine breakdowns occur with an inter-failure time of EXP(15000), and repair times were generated from EXP(500).
- Minor breakdowns occur with an interval generated from EXP(6000) for each machine, and repair time follows EXP(150).
- Buffer size, i.e., maximum number of available waiting jobs at each stage, was set to 200.

Due to a large number of operations and reentrant flows, the TFT-LCD manufacturing line has low system throughput, long flow time, and bad due date related performance measures. Therefore, their motive is to suggest new and practical real-time scheduling mechanism that can help to improve its system performances. Multiple performance measures are considered in this study. They are: (a) maximizing system throughput; (b) minimizing mean flow time; (c) minimizing mean tardiness; and (d) minimizing the number of tardy jobs.

3.1.7 Dispatching rules

Dispatching rules are used for selecting a job among those waiting in a queue at each stage when a machine becomes available. The dispatching rules tested in the case study are summarized below. Note that other rules can be added since the real-time mechanism is flexible in this respect.

- *FCFS* (first come first served): select an operation that arrived at the queue first.
- *SPT* (shortest processing time): select an operation with the shortest operation processing time, i.e., min p_j, where p_j denotes the processing time of operation j.
- *LPT* (longest processing time): select an operation with the longest operation processing time, i.e., max p_j.
- *LOR* (least operation remaining): select an operation with the least number of remaining operations, i.e., min o_j, where o_j denotes the

remaining operations of operation j (number of successor operations including itself).

- *MOR* (most operation remaining): select an operation with the largest number of remaining operations, i.e., max o_j.
- *LWR* (least work remaining): select the operation with the least remaining work, i.e., min r_j, where r_j denotes the remaining work of operation j (sum of processing times of the successor operations including itself).
- *MWR* (most work remaining): select the operation with the most remaining work, i.e., max r_j.
- *PWR* (processing time to work remaining): select an operation with the smallest ratio of the processing time to remaining work, i.e., min p_j/r_j.
- *POR* (processing time to operation remaining): select an operation with the smallest ratio of the processing time to remaining operations, i.e., min p_j/o_j.
- *EDD* (earliest due date): select an operation with the earliest due date, i.e., min d_j, where d_j denotes the due date of the job in which operation j is included.
- *SLACK* (minimum slack): select an operation with the minimum slack time, i.e., min $\{d_j - r_j - t\}$, where t is the current time.
- *MDD* (modified due date): select an operation with the minimum modified due date, where the modified due date of operation j is defined as max $\{d_j, t + r_j\}$.
- *S/RO* (slack per remaining operations): select an operation with the smallest ratio of slack time to the remaining operations, i.e., $(d_j - r_j - t)/o_j$.
- *S/RW* (slack per remaining work): select an operation with the smallest ratio of slack time to the remaining work, i.e., $(d_j - r_j - t)/r_j$.

3.1.8 Experimental design and results

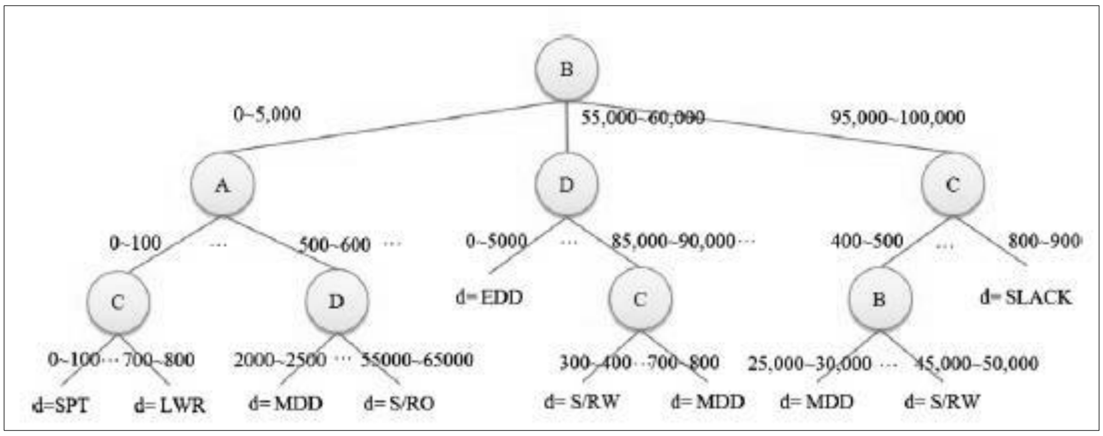


Fig. 3.4: A part of the decision tree used in the case study, Choi et al. (2010)

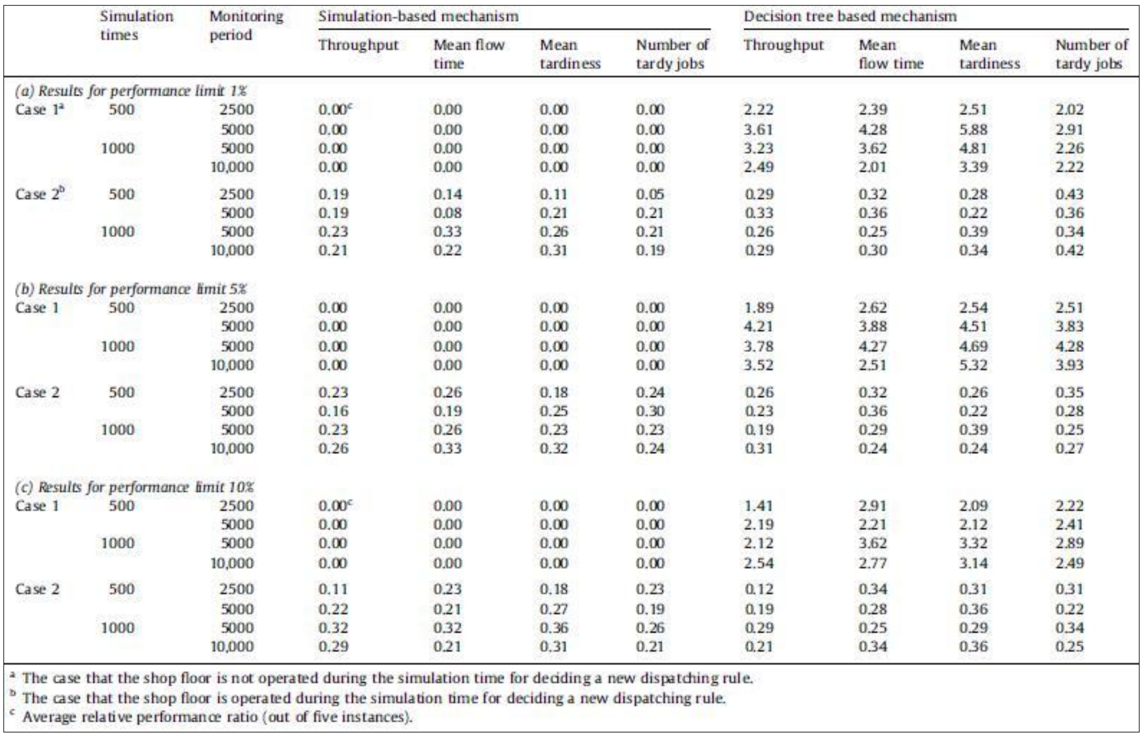
The main purpose of the test is to compare the decision tree based real-time scheduling mechanism (that eliminates the computational burdens of simulation runs for selecting dispatching rules) with the existing simulation-based one (that selects dispatching rules using time-consuming simulation results). As stated earlier, the performance measures considered in this study are system throughput, mean flow time, mean tardiness, and the number of tardy jobs. In this study, the data for constructing the decision tree were obtained from steady-state simulation runs because the system has no preliminary data. The two real-time scheduling mechanisms, together with the simulation model, were coded in C++ and the test was done on a workstation with an Intel Xeon processor operating at 3.2 GHz clock speed. The comparisons were done in two cases. The first case assumes that the shop floor is not operated during the simulation run time for deciding a new dispatching rule and hence the losses in system performances are not considered. In this case, the simulation based mechanism gives better results than the decision tree based one because the decision tree based one is an approximation of the simulation-based one. Nevertheless, the decision tree based mechanism has an inherent merit in that the simulation model needs not be required. On the other hand, in the second case, the shop floor is operated with the current dispatching rule during the simulation run time and hence the losses in system performances are explicitly considered. In the test, we performed the comparisons according to three levels of the performance limit in the scheduling strategy (1%, 5% and 10%). Recall that one of the scheduling strategies is that a new rule is selected if a certain performance value exceeds a predetermined performance limit at the end of each monitoring period. For each level of the performance limit, they performed five replications for each of the eight combinations for two levels for the simulation time for deciding a new dispatching rule in the simulation-based mechanism (500 and 1000) and three levels for the length of the periodic monitoring period (2500, 5000, and 10,000). The performance measure used is the relative performance ratio, which is defined as

$$100 \cdot \frac{C_a - C_{best}}{C_{best}}$$

for the minimization objectives (mean flow time, mean tardiness and the number of tardy jobs), and

$$100 \cdot \frac{C_{best} - C_a}{C_{best}}$$

for the maximization objective (system throughput), where C_a is the solution value obtained from real-time scheduling mechanism a and C_{best} is the better one of the two solution values.



	Simulation times	Monitoring period	Simulation-based mechanism				Decision tree based mechanism			
			Throughput	Mean flow time	Mean tardiness	Number of tardy jobs	Throughput	Mean flow time	Mean tardiness	Number of tardy jobs
(a) Results for performance limit 1%										
Case 1[a]	500	2500	0.00[c]	0.00	0.00	0.00	2.22	2.39	2.51	2.02
		5000	0.00	0.00	0.00	0.00	3.61	4.28	5.88	2.91
	1000	5000	0.00	0.00	0.00	0.00	3.23	3.62	4.81	2.26
		10,000	0.00	0.00	0.00	0.00	2.49	2.01	3.39	2.22
Case 2[b]	500	2500	0.19	0.14	0.11	0.05	0.29	0.32	0.28	0.43
		5000	0.19	0.08	0.21	0.21	0.33	0.36	0.22	0.36
	1000	5000	0.23	0.33	0.26	0.21	0.26	0.25	0.39	0.34
		10,000	0.21	0.22	0.31	0.19	0.29	0.30	0.34	0.42
(b) Results for performance limit 5%										
Case 1	500	2500	0.00	0.00	0.00	0.00	1.89	2.62	2.54	2.51
		5000	0.00	0.00	0.00	0.00	4.21	3.88	4.51	3.83
	1000	5000	0.00	0.00	0.00	0.00	3.78	4.27	4.69	4.28
		10,000	0.00	0.00	0.00	0.00	3.52	2.51	5.32	3.93
Case 2	500	2500	0.23	0.26	0.18	0.24	0.26	0.32	0.26	0.35
		5000	0.16	0.19	0.25	0.30	0.23	0.36	0.22	0.28
	1000	5000	0.23	0.26	0.23	0.23	0.19	0.29	0.39	0.25
		10,000	0.26	0.33	0.32	0.24	0.31	0.24	0.24	0.27
(c) Results for performance limit 10%										
Case 1	500	2500	0.00[c]	0.00	0.00	0.00	1.41	2.91	2.09	2.22
		5000	0.00	0.00	0.00	0.00	2.19	2.21	2.12	2.41
	1000	5000	0.00	0.00	0.00	0.00	2.12	3.62	3.32	2.89
		10,000	0.00	0.00	0.00	0.00	2.54	2.77	3.14	2.49
Case 2	500	2500	0.11	0.23	0.18	0.23	0.12	0.34	0.31	0.31
		5000	0.22	0.21	0.27	0.19	0.19	0.28	0.36	0.22
	1000	5000	0.32	0.32	0.36	0.26	0.29	0.25	0.29	0.34
		10,000	0.29	0.21	0.31	0.21	0.21	0.34	0.36	0.25

[a] The case that the shop floor is not operated during the simulation time for deciding a new dispatching rule.
[b] The case that the shop floor is operated during the simulation time for deciding a new dispatching rule.
[c] Average relative performance ratio (out of five instances).

Table 3.3: Results for the comparison test, Choi et al. (2010)

When constructing the decision tree using the preliminary simulation results, the conditional attributes were defined in the form of range instead of value. The resulting decision tree can be represented as Fig. 3.4 in which conditional attributes A, B, C and D denote the total number of remaining operations for the jobs in queue at each stage, the total processing time of remaining operations for the jobs in queue at each stage, the total number of remaining operations for the jobs being processed at each stage, and the total processing time of remaining operations for the jobs being processed at each stage, respectively. Test results on the two real-time scheduling mechanisms are summarized in Table 3.3. As can be seen in the table, the main result is that the differences in performances are not significantly large. (Recall that the decision tree based mechanism needs not require simulation runs.) In particular, for the second case in which the shop floor is operated with the current dispatching rule (before change) during the simulation time, there were no significant differences for all performance measures, which implies that the losses in system performances due to poor dispatching rules during the simulation run time are significant. Also, we found that the decision tree based mechanism may give better performances for some measures and parameter values. It was observed that the gaps between the two mechanisms get smaller as the performance limit gets increased (from 1% to 10%) because the current bad dispatching rule is used longer under larger performance limits. Therefore, we can see that the rule section mechanism plays an important role for immediate responses to changes in system states. In summary, we can argue that the decision tree based mechanism is worth to be considered for practical scheduling problems, especially, in the scheduling systems without preparing simulators.

3.1.9 Conclusion remarks

We considered the scheduling problem in reentrant hybrid flow shops that have a number of applications in various manufacturing and service systems. Unlike the existing theoretical approaches, they suggested a real-time scheduling mechanism in which a decision tree is used to select an appropriate dispatching rule so that the computational burden required for carrying out simulations can be eliminated. The decision tree based real-time scheduling mechanism was applied to a TFT-LCD manufacturing line, i.e., a typical reentrant hybrid flow shop, and the test results showed that it is competitive to the simulation-based one with respect to various performance measures such as system throughput, mean flow time, mean tardiness, and the number of tardy jobs. As a modification of the existing simulation-based real-time scheduling mechanism, this research can be extended in several directions. First, other algorithms to construct the decision tree may be used. In other words, it may be needed to construct more sophisticated decision trees. Second, more case studies that incorporate specific system characteristics are worth to be performed. Note that this work was supported by Brain Korea 21 Grant funded by Korean Government.

3.2 T. Sawik; *An Exact Approach for Batch Scheduling in Flexible Flow Lines with Limited Intermediate Buffers.* MATHEMATICAL AND COMPUTER MODELLING (2002) 36: 461-471

3.2.1 Introduction

First of all, a little nomenclature of the abbreviations that the author will use into the paper:

- System parameters
  - *G* : batch (part type), $g \in G = \{I,\ldots, v)$
  - *i* : processing stage, $i \in I = \{ 1,\ldots, m\}$
  - *j* : processor in stage i, $j \in J_i = \{1,\ldots, m_i\}$
  - *k* : part, $k \in K =\{1,\ldots, n\}$
- Input Parameters
  - b_g : size of batch g (number of parts of type g)
  - *m* : number of processing stages
  - m_t : number of parallel processors in stage t
  - *n* : total number of parts
  - r_{ig} : processing time in stage i of part type g

  - v : number of batches (part types)
  - K_g : subset of parts of type g
  - Q_{ifg} : large numbers not less than the schedule length
- Decision Variables
  - C_{max} : schedule length
  - c_{ik} : completion time of part k in stage i
  - d_{ik} : departure time of part k from stage i
  - x_{ijk} : if part k is assigned to processor $j \in J_i$, in stage $i \in I$; otherwise $x_{ijk} = 0$
  - y_{fg} : 1, if batch f precedes batch g; otherwise $y_{fg} = 0$

Now an introduction about the work and the case study is given. A flexible flow line (FFL) consists of several processing stages in series, separated by finite intermediate buffers, where each stage has one or more parallel identical machines. The line produces several different part types. Each part must be processed by at most one machine in each stage. A part which has completed processing on a machine in some stage is transferred either directly to an available machine in the next stage or to a buffer ahead of that stage. The limited intermediate buffers between the stages result in a *blocking scheduling* problem, (see McCormick et al.,1989) where a completed part may remain on a machine and block it until a downstream machine becomes available. This prevents another part from being processed on the blocked machine. A practical example of an FFL is an automated surface mount technology (SMT) line for printed wiring boards assembly, which includes three different processes in the following sequence: solder printing, component placement, and solder reflow. An example of an SMT line with parallel stations is shown in Figure 3.5. The line consists of a board loader, a solder printer, two parallel placement machines for small components, and two additional shuttles routing the board to the next available placement machine, one placement machine for fine pitch components, and a reflow oven. The assembly process is as follows: a tote of bare (preassembly) boards is brought to the beginning of the line, and a material loader loads each board separately on the conveyor. Each board is transported by the conveyor system through each processing stage in the line and then is stored again in a tote box. The loader and the tote box are used as the input and output buffers of the line. There are external buffers in front of and behind each placement machine, except the last one. In addition, every placement machine has its own internal input and output buffers of a fixed capacity.

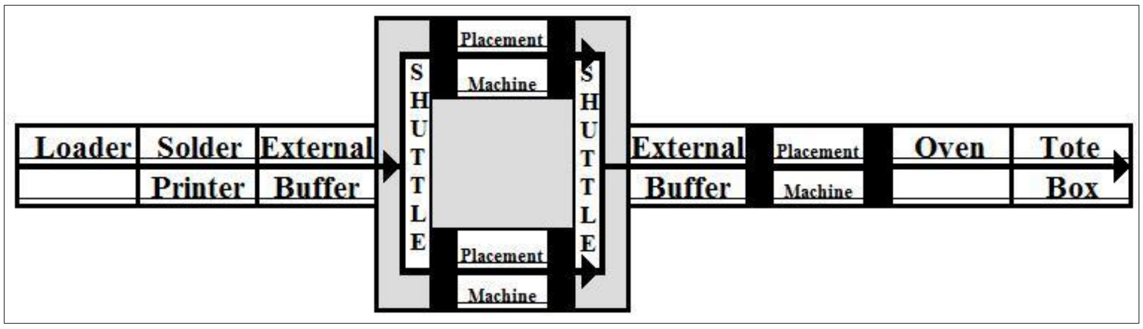


Fig. 3.5: An SMT line with parallel stations, Sawik (2002)

The objective of an FFL scheduling is to determine the detailed sequencing and timing of all processing tasks for each individual part, so as to maximize the line's productivity, which may be defined in terms of throughput or the schedule length (makespan) for a mix of part types. The problem of minimizing makespan in an FFL line is clearly NP-hard. An FFL line is a generalization of a multistage hybrid flowshop with parallel identical machines in each stage and unlimited intermediate buffers. Minimizing makespan in the hybrid flowshop is NP-hard, e.g., (Also, the m-machine flowshop with finite intermediate buffers is NP-hard even for m = 2). Furthermore, well solvable special cases such as two-machine flowshops with unlimited buffers or with no buffers are not directly applicable in the FFL environment. In practice, scheduling of an FFL is often based on daily demands and a simple approach to executing daily production plan is the use of batch scheduling, where parts of one type are processed consecutively. Since the batch sequencing problem in the two-machine flowshop with a finite intermediate buffer is NP-hard, minimizing makespan in the batch scheduling of a flexible flow line is NP-hard as well. In high-volume production, the production plan is often split into several identical sets of smaller batches of parts that are scheduled repeatedly. The smallest possible set of parts in the same proportion as the daily part mix requirements is called the minimal part set (MPS) (Wittrock, 1985). Research on scheduling algorithms for FFL is mostly restricted to heuristics which seek good solutions within reasonable computation times (Kim et al., 1996). This paper, however, provides the reader with an exact approach based on a mixed integer programming formulation of the FFL scheduling problem. The formulation can be applied for constructing optimal batch schedules for small size batches of different part types (e.g., for MPS) and for various FFL configurations by using commercially available software for mixed integer programming. This has been illustrated in the paper with numerical examples that have been modeled after real-world SMT lines using an advanced algebraic modeling language AMPL and the CPLEX solver. The paper is organized as follows. In the next section, a mixed integer programming formulation is presented for batch scheduling in a flexible flow line with machine blocking. Numerical examples modeled after real-world SMT lines and some computational results are provided and conclusions are given in the last section.

3.2.2 Mixed integer program for batch scheduling in a flexible flow line with machine blocking

In this section, a mixed integer programming model is presented for batch scheduling in a flexible flow line with limited intermediate buffers. A unified modeling approach is adopted with the buffers viewed as machines with zero processing times. As a result, the scheduling problem with buffers can be converted into one with no buffers but with blocking, see McCormick et al. 1989. The blocking time of a machine with zero processing time denotes part waiting time in the buffer represented by that machine. We assume that each part must be processed in all stages, including the buffer stages. However, zero blocking time in a buffer stage indicates that the corresponding part does not need to wait in the buffer. Let us note that for each buffer stage part completion time is equal to its departure time from the previous stage since the processing time is zero.

Notation used to formulate the problem is shown in the nomenclature, where buffers and machines are referred to as processors.

The flexible flow line under study consists of m processing stages in series. Each stage i (i = 1,...,m) is made up of $m_i \geq 1$ identical parallel processors. Let J_i, be the circular set of indices of parallel processors in stage i. The system produces various types of parts. Let G = {1,…,v}, K = {1,…,n} and K_g = $\{\sum_{f\in G:f\leq g-1} b_f + 1,\ldots, \sum_{f\in G:f\leq g-1} b_f + b_g\}$ be the ordered sets of indices, respectively, of all batches of parts, all individual parts, and all parts of type g ∈ G. (b_g , $n = \sum_{g=1}^{v} b_g$ and *v*, denote respectively the number of parts of type g, the total number of parts, and the number of batches in the schedule.)

All parts are scheduled in batches of parts of the same type and within the batch individual parts are processed consecutively part-by-part. No setups are required between different parts or different batches of parts. Each part must be processed without preemption on exactly one processor in each of the stages sequentially. That is, each part must be processed in Stage *1* through Stage *m* in that order. The order of processing the parts in every stage is identical and determined by an input sequence in which the parts enter the line, i.e., a so-called permutation flowshop is considered.

For every part *k,* denote by c_{ik} its completion time in each Stage i, and by d_{ik} the departure time from stage *i*).

Let $r_{ig} \geq 0$, be the processing time in Stage *i* of each part type $g \in G$. Processing without preemption indicates that part $k \in K_g$ completed in Stage *i* at time c_{ik} starts its processing in that stage at time $c_{ik} - r_{ig}$. Part $k \in K_g$ completed in Stage *i* at time c_{ik} departs at time $d_{ik} \geq c_{ik}$ to an available processor in the next Stage *i + 1*. If at time c_{ik} all m_{i+1} processors in Stage *i + 1*are occupied, then the processor in Stage *i* is blocked by part *k* until time $d_{ik} = c_{i+1k} - r_{i+1g}$ when part $k \in K_g$ starts processing on an available processor in Stage *i + 1*.

The objective is to determine an input sequence of batches and an assignment of parts to processors in each stage over a scheduling horizon to complete all the parts in minimum time, that is, to minimize the makespan $C_{max} = \max_{k\in K} (C_{mk})$, where C_{mk} denotes the completion time of part *k* in the last stage *m*. The mixed integer program for batch scheduling in a flexible fiow line with finite in-process buffers is presented below.

Minimize:

$$C_{max} \quad (1)$$

Subject to:

- *part assignment constraints*

$$\sum_{j \in J_i} x_{ijk} = 1;\ i \in I,\ k \in K,\ (2)$$

$$x_{i,next(j \in J_i),k+1};\ i \in I,\ j \in J_i,\ g \in G,\ k \in K_g : k < last(K_g),\ m_i > 1;\ (3)$$

- *part completion constraints*

$$c_{1k} \geq r_{1g};\ g \in G,\ k \in K_g\ (4)$$

$$c_{ik} - c_{i-1k} \geq r_{ig};\ i \in I,\ g \in G,\ k \in K_g : i > 1;\ (5)$$

- *part departure constraints*

$$c_{ik} \leq d_{ik};\ i \in I,\ k \in K : i < m,\ (6)$$

$$c_{mk} = d_{mk};\ k \in K;\ (7)$$

- *part noninterference constraints*

$$c_{1k} + Q_{ifg}\,(2 + y_{fg} - x_{ijk} - x_{ijl}) \geq d_{il} + r_{1f}$$

$$i \in I,\ j \in J_i,\ f,g \in G,\ k \in K_f,\ l \in K_g : f < g;\ (8)$$

$$c_{1l} + Q_{igf}\,(3 - y_{fg} - x_{ijk} - x_{ijl}) \geq d_{ik} + r_{1g}$$

$$i \in I,\ j \in J_i,\ f,g \in G,\ k \in K_f,\ l \in K_g : f < g;\ \ (9)$$

- *buffering constraints*

$$c_{ik} = d_{i-1k} + r_{ig};\ i \in I,\ g \in G,\ k \in K_g : i > 1;\ (10)$$

- *maximum completion time constraints*

$$c_{mk} \leq C_{max};\ \ k \in K,\ (11)$$

$$d_{ik} + \sum_{h \in I, h > i} r_{hg} \leq C_{max};\ i \in I,\ g \in G,\ k \in K_g : i < m\ ,\ (12)$$

$$c_{ml} - c_{il} \leq C_{max} - \sum_{f \in G, f < g} b_f\, r_{1f}\, y_{fg} - \sum_{f \in G, f > g} b_f\, r_{1f} (1 - y_{gf}) -$$

$$(l - \sum_{f=1}^{g-1} b_f) r_{1g} - (\sum_{f=1}^{g} b_f - l) r_{mg} - \sum_{f\in G, f<g} b_f r_{mf}(1 - y_{fg}) -$$

$$\sum_{f\in G, f>g} b_f r_{mf}\, y_{gf};\quad g \in G,\ l \in K_g : m_1 = 1,\ m_m = 1;\quad (13)$$

- *batch processing constraints*

$$c_{ik+m} \ge d_{ik} + r_{ig};\quad i \in I,\ g \in G,\ k \in K_g : k + m_i < last(K_g),\ m_i > 1,\ (14)$$

$$c_{ik+1} \ge c_{ik};\quad i \in I,\ ,\ g \in G,\ k \in K_g : k < last(K_g),\ m_i > 1,\ (15)$$

$$c_{ik+1} \ge d_{ik} + r_{ig};\ i \in I,\ g \in G,\ k \in K_g : k < last(K_g),\ m_i = 1;\ (16)$$

- *variable elimination constraints*

$$f,g \in G : f \ge g;\ (17)$$

- *variable non-negativity and integrality constraints*

$$i \in I,\ k \in K,\ (18)$$

$$i \in I,\ k \in K,\ (19)$$

$$i \in I,\ j \in J_i,\ k \in K,\ (20)$$

$$f,g \in G.\ (21)$$

The objective function (1) represents the schedule length to be minimized. Assignment constraint (2) ensures that in every stage each part is assigned to exactly one processor, and (3)assigns successive parts of one type alternatively to different parallel processors (*next(j,J$_i$)* is the next processor after *j* $\in$ *J,* in the circular set *J$_i$* of parallel processors at Stage *i*). Constraint (4) ensures that each part is processed in the first stage, and (5) guarantees that it is also processed in all downstream stages. Constraint (6) indicates that each part cannot be departed from a stage until it is completed in this stage, and equation (7) ensures that each part leaves the line as soon as it is completed in the last stage. Constraints (8) and (9) are part noninterference constraints. No two parts can be performed on the same processor simultaneously. For a given sequence of parts, only one constraint of (8) and (9) is active, and only if both parts *k* and *l* are assigned to the same processor. Equation (10) indicates that processing of each part in every stage starts immediately after its departure from the previous stage. Constraint (11) defines the maximum completion time of all parts. Constraint (12) relates part departure times to makespan directly. Every part must be departed from a stage sufficiently early in order to have all of its remaining tasks completed within the remaining processing time. Constraint (13) ensures that each part is processed within the time interval remaining after processing of all preceding parts and before processing of all succeeding parts. Flow time $c_{ml} - (c_{1l} - r_{1g})$ of each part $l \in K_g$ cannot be greater than the makespan $C_{\max}$ minus sum of processing times of all preceding parts in the first stage

$$\sum_{f\in G, fh<gh} b_f r_{1f} y_{fg} + \sum_{f\in G, fh>gh} b_f r_{1f} (1 - y_{gf}) + \\ (l - 1 - \sum_{f=1}^{g-1} b_f) r_{1g},$$

and sum of processing times of all succeeding parts in the last stage

$$(\sum_{f=1}^{g} b_f - l) r_{mg} + \sum_{f\in G, f<g} b_f r_{mf} (1 - y_{fg}) - \sum_{f\in G, f>g} b_f r_{mf} y_{gf}.$$

Constraint (13) is valid only for the line that begins and ends with a single processor, which is typical for SMT lines. Batch processing constraints (14),(15) along with (3) ensure that parts of one type are processed consecutively in each stage with parallel processors, whereas consecutive processing of identical parts in each stage with a single processor is imposed by (16). Parameter Q_{ifg} in (8) and (9) is a large number not less than the schedule length, determined for Stage i when batch f precedes batch g. Q_{ifg} is calculated as below, where UB is an upper bound on the schedule length.

$$Q_{ifg} = UB - \sum_{h\in I: h<i} r_{hf} - \sum_{h\in I: h>i} r_{hg}; \; i \in I; \; f,g \in G, \quad (22)$$

$$UB = \frac{\sum_{i\in I} \sum_{g\in G} b_g r_{ig}}{m_i}$$

The mixed integer program includes various cutting constraints exploiting special FFL configurations (e.g., constraints (12),(13)) and some properties of batch processing on parallel processors ((3), (14), and (15)) and on a single processor (16). The cutting constraints may have a great impact on reducing computational effort required to find the optimal solution. The model proposed for batch scheduling in flexible flow lines with limited intermediate buffers is a general formulation and includes various special cases. For example, if $m_i = 1, \forall i \in I : \sum_{g \in G} r_{ig} > 0$, the model can be applied for batch scheduling in a flexible flow line with no buffers. If completion and departure times are equal for each processing stage and part, i.e., $c_{ik} = d_{ik}$, $\forall i \in I, k \in K,$ the batch scheduling problem in a hybrid flowshop with unlimited buffers can be considered.

3.2.3 Numerical examples

In this section, numerical examples are presented, and some computational results are reported to illustrate possible applications of the mixed integer programming approach. The examples are modeled after real-world SMT lines. The assembly schedules for the examples were calculated on a Compaq Presario 1830 laptop with Pentium III, 450MHz using AMPL modeling language and CPLEX v.7.1 solver.

EXAMPLE 1. FACTORY WITH SINGLE STATIONS.
The SMT line configuration for Example 1 is shown in Figure 3.6. The line consists of a loader, screen printer, four placement machines, and a vision inspection machine, in series separated by intermediate buffers. The line represents a typical low-volume, medium-variety production system. For the industry scenario that was studied, 13 different board types are assembled in small size batches. A daily production order consists of at most four different board types assembled in the line.
The input data for Example 1 were prepared considering the daily production of the line over a one month horizon. Table 3.4 lists the processing times for boards, and Table 3.5 presents the input data for selected problem instances that represent five daily production orders. The

characteristics of mixed integer programs for the example and the solution results are summarized in Table 3.6.

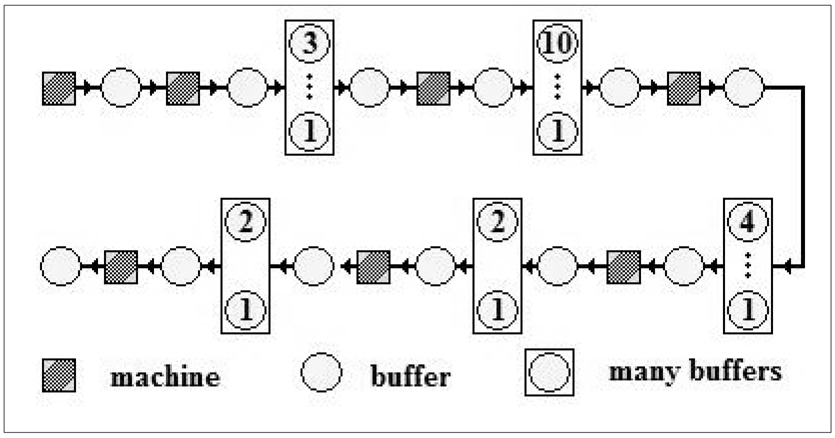


Fig. 3.6: Factory with single stations, Sawik (2002)

Board Type	*Processing stage*						
	1	*3*	*7*	*11*	*15*	*19*	*23*
1	20	25	133	45	38	62	45
2	20	25	155	156	28	58	50
3	20	25	67	56	36	35	45
4	20	25	93	95	-	51	40
5	20	25	76	111	41	63	50
6	20	25	87	93	52	48	45
7	20	25	34	78	92	55	45
8	20	25	66	28	34	-	30
9	20	25	141	90	49	-	40
10	20	25	86	83	56	22	45
11	20	25	98	84	36	43	45
12	20	25	176	175	76	65	50
13	20	25	-	17	67	28	45

Table 3.4: EXAMPLE 1. Processing times in seconds, Sawik (2002)

Problem no.	*Daily Mix*							
	Board Type	*Batch Size*	*Board Type*	*Batch Size*	*Board Type*	*Batch Size*	*Board Type*	*Batch Size*
1	7	13	9	6	-	-	-	-
2	2	23	9	1	-	-	-	-
3	7	1	11	33	-	-	-	-
4	5	17	7	1	8	11	9	1
5	1	21	4	1	7	2	10	7

Table 3.5: EXAMPLE 1. Input data for selected problems, Sawik (2002)

Problem	*Var.*	*Bin.*	*Cons.*	*Nonz.*	*LB*	*C^{*}_{max}*	*Nodes*	*CPU^{**}*
1	1085	590	7289	30771	1722	1722	0	5,5
2	1370	745	3364	11926	3953	3967	0	1
3	1940	1055	4824	17096	3521	3521	0	1,9
4	1717	936	20939	91916	2506	2568	19	52
5	1774	967	20158	88157	3577	3577	18	28

Table 3.6: EXAMPLE 1. Computational results, Sawik (2002)
****Optimal makespan***
*****CPU time for proving optimality***

The size of the mixed integer programming models for the example problems is represented by the total number of variables, *Var.*, number of binary variables, *Bin.*, number of constraints, *Cons.*, and number of nonzero coefficients, *Nonz.*, in the constraint matrix. The last four columns of Table 3.6 present the lower bound *LB* on the makespan, the optimal makespan C_{max} , the node number in the branch-and-bound tree at which the optimal solution was found, and *CPU* time in seconds required to prove optimality of the solution. In all cases, the time required to find the optimal solution was much smaller than that required to prove optimality. The lower bound was calculated as below

$$\mathrm{LB} = \max\left\{\frac{\sum_{g\in G} b_g r_{ig}}{m_i} + \min_{g\in G}\left(\sum_{h\in I, h<i} r_{hg}\right) + \min_{g\in G}\left(\sum_{h\in I, h>1} r_{hg}\right)\right\}$$

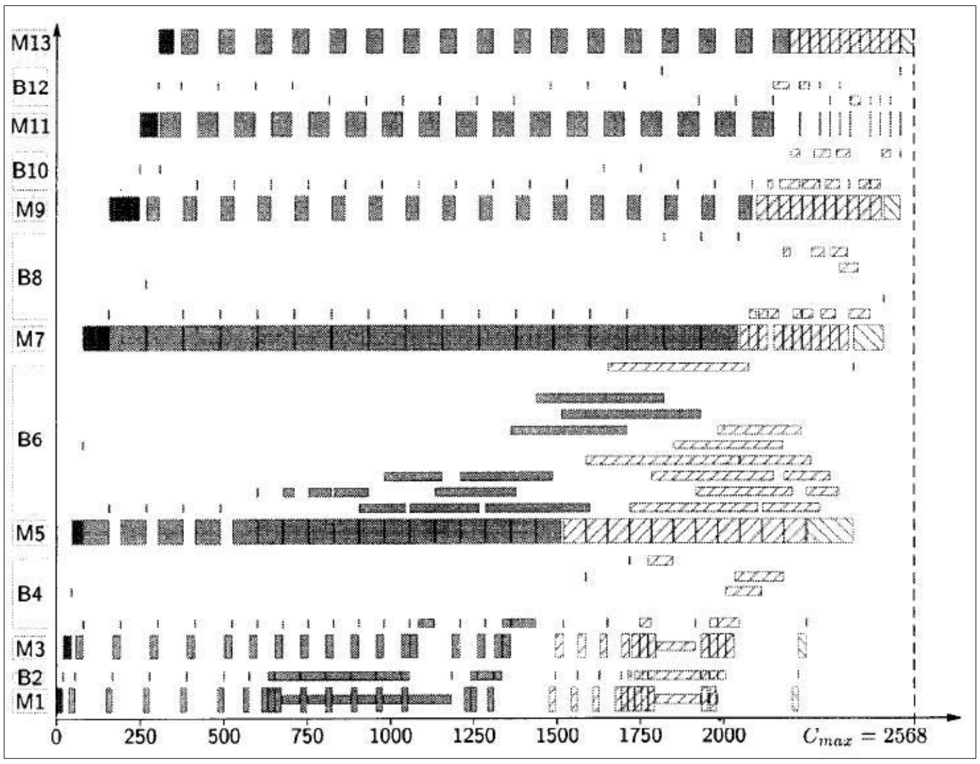


Fig. 3.7: Batch schedule for SMT line with single stations, Sawik (2002)

Figure 3.7 shows a Gantt chart with the optimal batch schedule for Problem 4, where B stands for buffer and M stands for machine for board loading, screen printing, component placement, or vision inspection. Buffering or machine blocking is indicated with a narrow bar. The optimal input sequence of board types is 7,5,8,9, and the optimal makespan C_{max} = 2568.

EXAMPLE 2. FACTORY WITH PARALLEL STATIONS

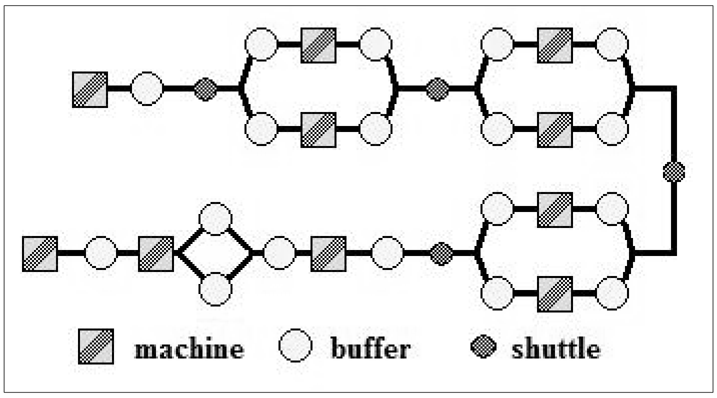


Fig 3.8: Factory with parallel stations, Sawik 2002

The SMT line configuration for Example 2 is shown in Figure 3.8. The line consists of a screen printer, three sets of two parallel placement machines and four shuttles routing the boards to the next available placement machine, a vision inspection machine and a single placement machine, in series separated by intermediate buffers. The line represents a typical high-volume, low-variety production system, in which six different board types are produced in medium to large size batches. A daily production order consists of at most four different board types assembled in the line. Table 4 lists the processing times for boards, and Table 5 presents the input data for selected problem instances that represent five daily production orders and the corresponding minimum part sets. The MPS production requirements represent 1/40th, 1/40th, 1/30th, 1/100th, and 1/40th of the actual daily production order, respectively, for Problems 1, 2, 3, 4, and 5.

Board Type	*Processing stage*						
	1	*5*	*9*	*13*	*17*	*19*	*21*
1	22	207	213	204	80	40	62
2	22	208	220	204	80	40	62
3	22	207	224	191	80	40	62
4	22	207	213	204	80	40	62
5	22	207	220	204	80	40	62
6	22	184	196	199	80	40	62

Table 3.7: EXAMPLE 2. Processing times in seconds, Sawik (2002)

Problem no.	*Daily Mix/ MPS*							
	Board Type	*Batch Size*	*Board Type*	*Batch Size*	*Board Type*	*Batch Size*	*Board Type*	*Batch Size*
1	3	240/6	4	200/5	5	480/12	-	-
2	1	80/2	2	120/3	3	240/6	5	480/12
3	1	180/6	2	210/7	3	510/17	-	-
4	3	300/3	4	400/4	5	500/5	-	-
5	3	1080/27	6	400/10	-	-	-	-

Table 3.8: EXAMPLE 2. Input data for selected problems, Sawik (2002)

Problem	Var.	Bin.	Cons.	Nonz.	LB	C^*_{max}	Nodes	CPU**
1	1269	670	12434	54674	3127	3233	17	41
2	1272	673	12799	56519	3137	3247	100	1400
3	1654	873	19674	87634	3915	3993	40	130
4	664	351	3950	16622	1914	1992	80	11
5	2037	1074	20758	91137	4583	4695	0	72

Table 3.9: EXAMPLE 2. Computational results, Sawik (2002)
****Optimal makespan***
*****CPU time for proving optimality***

The characteristics of mixed integer programs for the MPS problems and the solution results are summarized in Table 3.9. The last four columns of Table 3.9 present the lower bound LB, on makespan, the optirnal makespan C_{max} the node number in the branch-and-bound tree at which the optimal solution was found, and CPU time in seconds required to prove optimality of the solution. In all cases, the time required to find the optimal solution was much smaller than that required to prove optimality. Figure 3.9 shows a Gantt chart with the optimal batch schedule obtained for Problem 4, where B stands for buffer and M stands for machine for screen printing, component placement, or vision inspection. The input sequence of board types is 4,5,3, and the makespan C_{max} = 1992.

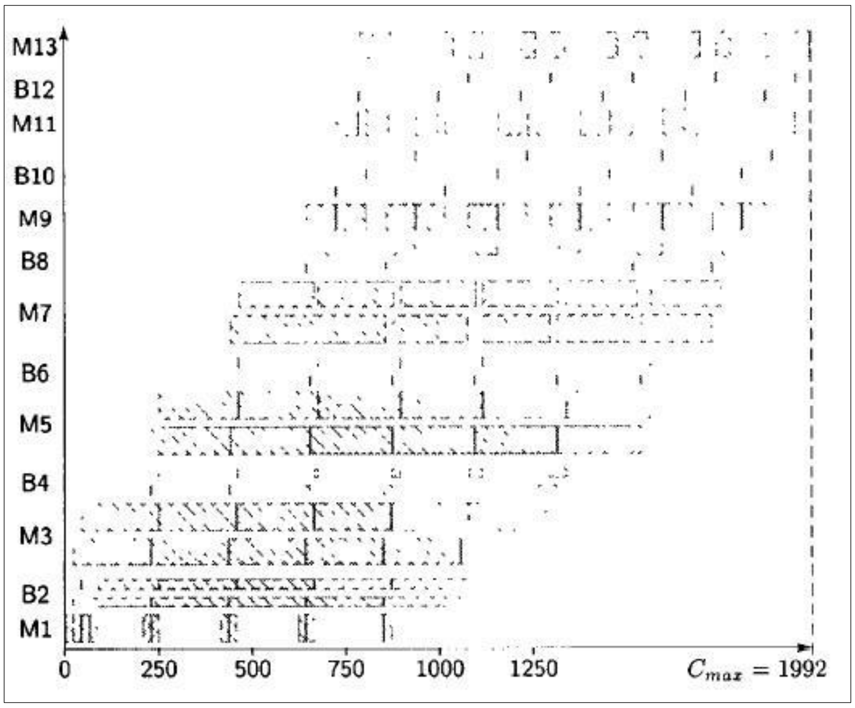


Fig 3.9: Batch schedule for SMT line with parallel stations, 2002

Experiments with various features of the CPLEX solver to speed up the solution process have indicated that the best results are obtained with a nearly depth-first branch and bound strategy for node selection where limited backtracking is allowed. *(Note that the quality is low because the image comes from the original paper and haven't been modified, in order to respect Sawik's work and to preserve the accuracy.)*

3.2.4 Conclusions

This paper has presented an exact approach for batch scheduling in flexible flow lines with limited intermediate buffers. The approach based on a mixed integer programming formulation is capable of optimal scheduling of batches of different part types by using commercially available software for discrete programming. The mathematical formulation includes various cutting constraints exploiting special FFL configurations and some properties of batch processing on parallel machines. The cutting constraints have an impact on reducing computational effort required to find the optimal solution. Nevertheless, the CPU time required to find proven optimal schedules for realistic large size problems still can be very high. The computation time can be further reduced by introducing a specific MPS scheduling mode. The proposed approach can be applied to a variety of different real-world flexible flow line configurations and production scenarios with only small modifications to the constraint formulations and input data definitions. The proven optimal solutions that can obtained for small size problems can also be used to evaluate the performance of various heuristics for FFL batch scheduling. Note that this work has been partially supported by AGH and KBN (Poland) and by the Motorola Advanced Technology Center (U.S.A.).

Chapter 4:
Conclusions and thanks

After a very long work, this should be easy, but the more I think about it, the more it seems to increase its difficult. Firstly I want to thank Dr. Faccio and Eng. Zanin for their help and support in my work, and for the long time they dedicated to me whenever I needed. Then I thanks all other teachers of University of Padua, because all the courses are and will be a part of my culture. But, despite this incredible people who accompany me through my university life there are other people who helped me so much during my free time and during my homework and studies, of course I'm speaking about friends, they are always kind and make my free time easy and light day by day. They are not the only to support me, i took them to the end because they are the most important: my beloved family. They all want me to keep on and to inform about my results, my exams, and never putting pressure on me. This is the way I would reward and repay them for everything. All the people I thanked will be always in my heart, but everyone should remember that when you arrive to one goal, this is a new beginning, and another goal will come, so I will need your love and support again and again.

Niccolò

Bibliography

- ✓ Alaykýran K., Engin O., Döyen A., 2007. *Using ant colony optimization to solve hybrid flow shop scheduling problems.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY 35: 541–550.
- ✓ Amin-Naseri M. R., Beheshti-Nia M. A., 2009. *Hybrid flow shop scheduling with parallel batching.* INTERNATIONAL JOURNAL OF PRODUCTION ECONOMICS 117: 185–196.
- ✓ Burns A., Wellings A. J., 1990*. Real-time systems and Programming Languages, 1st edition.* ADDISON-WESLEY, READING, MA.
- ✓ Caricato P., Grieco A., Pacella M., 2001. *Optimal scheduling of non-independent jobs on fspm: a simulation-based hybrid algorithm.* PROCEEDINGS OF THE ANNUAL CONFERENCE OF THE ITALIAN SOCIETY FOR COMPUTER SIMULATION (ISCS), NAPLES: 193-200.
- ✓ Caricato P., Grieco A., Serino D., 2005. *Tsp-based scheduling in a batch-wise hybrid flow-shop.* ROBOTICS AND COMPUTER-INTEGRATED MANUFACTURING 23: 234–241.
- ✓ Carlier J., Neron E., 2000. *An exact method for solving the multiprocessor flow-shop.* REVUE D'AUTOMATIQUE, D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE (RAI-RO), 34(1): 1–25.
- ✓ Central Intelligence Agency, 2007. *Central Intelligence Agency's World factbook.* www.cia.gov.
- ✓ Cheng T. C. E., Wang G., 1999. *Scheduling the fabrication and assembly of components in a two-machine flowshop.* IIE TRANSACTION 31: 135-143.
- ✓ Choi H.-S., Lee D.-H., 2009. *Scheduling algorithms to minimize the number of tardy jobs in two-stage hybrid flow shops.* COMPUTERS & INDUSTRIAL ENGINEERING 56: 113–120.
- ✓ Choi H.-S., Kim J.-S., Lee D.-H., 2010. *Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a TFT-LCD line.* EXPERT SYSTEMS WITH APPLICATIONS (2010) (ARTICLE IN PRESS) (DOI:10.1016/J.ESWA.2010.08.139).

- ✓ Dasgupta D., Attoh-Okine N., 1997. *Immunity-based systems: a survey.* PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON COMPUTATIONAL CYBERNETIC SIMULATION 1: 369:374.
- ✓ Deneubourg J.-L., Aron S., Goss S., Pasteels J.-M., 1990. *The self-organizing exploratory pattern of the Argentine ant.* JOURNAL OF INSECT BEHAVIOR 3: 159 1990.
- ✓ Dominic P. D. D., Kaliyamoorthy S., Saravana Kumar M., 2004. *Efficient dispatching rules for dynamic job shop scheduling.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY 24: 70-75.
- ✓ Dorigo M., 1992. *Optimization, Learning and Natural Algorithms. (Phd Thesis).* POLITECNICO DI MILANO, ITALIE.
- ✓ Engin O., Doyen A., 2004. *A new approach to solve hybrid flow shop scheduling problems by artificial immune system.* FUTURE GENERATION COMPUTER SYSTEMS, 20(6): 1083–1095.
- ✓ Etmaghraby S. E., Kamoub R. E., 1997. *Production control in hybrid flowshops: an example from textile manufacturing.* THE PLANNING AND SCHEDULING OF PRODUCTION SYSTEMS (6) CHAPMAN & HALL, UK.
- ✓ Ferguson T. S., 2000. *Linear Programming: a concise introduction*. T. S. FERGUSON'S WEBSITE (http://www.math.ucla.edu.com/~tom).
- ✓ Garcia J. M., Lozano S., 2005. *Production and delivery scheduling problem with time windows.* COMPUTERS & INDUSTRIAL ENGINEERING 48: 733–742.
- ✓ Garey M. R., Johnson D. S., 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. FREEMAN.
- ✓ Gholami M., Zandieh M., Alem-Tabriz A., 2008. *Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY 42: 189–201.
- ✓ Goldberg D. E. , 1989. *Genetic algorithms in search, optimization and machine learning.* ADDISON-WESLEY, READING, MA.
- ✓ Goss S., Aron S., Deneubourg J.-L., Pasteels J.-M., 1989. *The self-organized exploratory pattern of the Argentine ant.* NATURWISSENSCHAFTEN 76: 579-581.

- ✓ Gupta J. N. D., Darrow W.P., 1986. *Schedules for a two-stage hybrid flowshop with parallel machines at the second stage.* INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH 29: 1489-1502.
- ✓ Gupta J. N. D., 1988. *Two-stage hybrid flow shop scheduling problem.* JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY 39: 359–364.
- ✓ Gutin G., Punnen A., 2002. *The traveling salesman problem and its variation.* DORDRECHT: KLUWER.
- ✓ Han S. W., 2008. *Rough set based decision tree with static and dynamic entities.* PH.D. DISSERTATION. SEOUL, SOUTH KOREA: HANYANG UNIVERSITY.
- ✓ Haouari M. , Hidri L., Gharbi A., 2006. *Optimal scheduling of a two-stage hybrid flow shop.* MATHEMATICAL METHODS OF OPERATION RESEARCH 64: 107–124.
- ✓ Holland J. H., 1975. *Adaptation in Natural and Artificial Systems.* UNIVERSITY OF MICHIGAN PRESS.
- ✓ Ishibuchi H., Yoshida T., Murata T., 2003. *Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling*. IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION 7: 204–23.
- ✓ J. Dréo, 2006. *Choix du plus court chemin par une colonie de fourmi.* J. DRÉO'S WEBSITE (http://www.nojhan.net/pro).
- ✓ Jeong K.-C., Kim Y.D., 1998. *A real-time scheduling mechanism for a flexible manufacturing system: Using simulation and dispatching rules.* INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH 36: 2609–2626.
- ✓ Jiao L., Wang L., 2000. *Novel genetic algorithm based on immunity.* IEEE TRANSACTION SYSTEM MANUFACTURING CYBERNETIC, PART A 30(5): 552–561.
- ✓ Jolai F., Sheikh S., Rabbani M. , Karimi B. , 2008. *A genetic algorithm for solving no-wait flexible flow lines with due window and job rejection.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY 42: 523–532.
- ✓ Jouglet A., Oguz C., Sevaux M., 2009. *Hybrid llow-shop: a memetic algorithm using constraint-based scheduling for efficient search.*

JOURNAL OF MATHEMATICAL MODELLING AND ALGORITHMS 8: 271–292.

- ✓ Jungwattanakit J., Reodecha M., Chaovalitwongse P., Werner F., 2007. *Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY 37: 354–370.
- ✓ Kahraman C., Engin O., Kaya I., Ozturk R. E., 2010. *Multiprocessor task scheduling in multistage hybrid flow-shops: a parallel greedy algorithm approach.* APPLIED SOFT COMPUTING 10: 1293–1300.
- ✓ Khalouli S., F. Ghedjati, A. Hamzaoui, 2010. *A meta-heuristic approach to solve a JIT scheduling problem in hybrid flow shop.* ENGINEERING APPLICATIONS OF ARTIFICAL INTELLIGENCE 23: 765–771.
- ✓ Kim Y.-D., Lim H.-G., Park M.-W., 1996. *Search heuristics for a Flowshop scheduling problem in a printed circuit board assembly process.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH 91: 124-143.
- ✓ Kirkpatrick S., Gelatt Jr. C. D., Vecchi M. P., 1983. *Optimization by Simulated Annealing.* SCIENCE 220 (4598): 671–680.
- ✓ Kolisch R., Sprecher A., Drexl A., 1992. *Characterization and generation of a general class of resource-constrained project scheduling problems.* TECHNICAL REPORT 301, MANUSKRIPTE AUS DEN INSTITUTEN FUR BETRIEBSWIRTSCHAFTSLEHRE DER UNIVERSITAT KIEL.
- ✓ Koulamas C., Kyparisis G. J., 2007. *A note on the two-stage assembly flow shop scheduling problem with uniform parallel machines.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH 182: 945–951.
- ✓ Kurz M. E., Askin R. G., 2003. *Comparing scheduling rules for flexible flow lines.* INTERNATIONAL JOURNAL OF PRODUCTION ECONOMY, 85: 371–388.
- ✓ Kurz M. E., Askin R. G., 2004. *Scheduling flexible flow lines with sequence-dependent setup times.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH, 159(1): 66–82.

- ✓ Land A. H., Doig G., 1960. *An automatic method of solving discrete programming problems.* ECONOMETRICA 28 (3): 497-520.
- ✓ Lin H.-T., Liao C.-J., 2003. *A case study in a two-stage hybrid flow shop with setup time and dedicated machines.* INTERNATIONAL JOURNAL OF PRODUCTION ECONOMICS 86: 133–143.
- ✓ Linn R., Zhang W., 1999. *Hybrid flow shop scheduling: a survey.* COMPUTERS & INDUSTRIAL ENGINEERING, 37(1): 57–61.
- ✓ Liu, H. J., Dong, J., 1996. *Dispatching rules selection using artificial neural networks for dynamic planning and scheduling.* JOURNAL OF INTELLIGENT MANUFACTURING, 7: 243–250.
- ✓ Loukil T., Teghem J., Fortemps P., 2007. *A multi-objective production scheduling case study solved by simulated annealing.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH, 179: 709–722.
- ✓ McCormick S. T., Pinedo M. L., Shenker S., Wolf B., 1989. *Sequencing in an assembly line with blocking to minimize cycle time.* OPERATIONS RESEARCH 37: 925-936.
- ✓ Mirsanei H. S., Zandieh M., Moayed M. J., Khabbazi M. R., 2010. *A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times.* JOURNAL OF INTELLIGENT MANUFACTURING, 18 JANUARY (DOI 10.1007/S10845-009-0373-8).
- ✓ Mladenovic N., Hansen P., 1997. *Variable neighborhood search.* COMPUTERS & OPERATIONS RESEARCH 24(11): 1097-1100.
- ✓ Morton T. E., Pentico D. W., 1993. *Heuristic Scheduling Systems.* WILEY, NEW YORK.
- ✓ Mousavi S. M., Zandieh M., Amiri M., 2010. *An efficient bi-objective heuristic for scheduling of hybrid flow shops.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, 27 OCTOBER (DOI 10.1007/S00170-010-2930-X).
- ✓ Naderi B., Zandieh M., Roshanaei V., 2008. *Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY 41:1186–1198.

- ✓ Nawaz M., Enscore J. E., Ham I., 1983. *A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem.* THE INTERNATIONAL JOURNAL OF MANAGEMENT SCIENCE 11: 91–95.
- ✓ Neron E., Baptiste P., Gupta J. N. D., 2001. *Solving hybrid flow shop problem using energetic reasoning and global operations.* OMEGA 29(6): 501–511.
- ✓ Nowicki E., 1999. *The permutation flow shop with buffers: a tabu search approach.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH 116: 205–19.
- ✓ Oguz C., Zinder Y., Do V. H., Janiak A., Lichtenstein M., 2004. *Hybrid flow shop scheduling problems with multiprocessor task systems.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH, 152: 115–131.
- ✓ Oguz C., Erkan M., 2005. *A genetic algorithm for hybrid flow shop scheduling with multiprocessor tasks.* JOURNAL OF SCHEDULING 8: 323–351.
- ✓ Oguz C., 2006. *Data for Hybrid Flow-shop Scheduling with Multiprocessor Tasks.* C. OGUZ'S WEBSITE (http://home.ku.edu.tr/~coguz).
- ✓ Pinedo M., 1995. *Scheduling theory, algorithms, and systems. 1st edition.* PRENTICE HALL, ENGLEWOOD CLIFFS, NJ.
- ✓ Pinedo M., 2002. *Scheduling Theory, Algorithms, and System. 2nd edition.* PRENTICE HALL, ENGLEWOOD CLIFFS, NJ.
- ✓ Portmann M.-C., Vignier A., Dardilhac D., Dezalay D., 1998. *Branch and bound crossed with GA to solve hybrid flow shops.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH, 107: 389–400.
- ✓ Prandtstetter M., Raidl G. R., 2007. *An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH, 191(3): 1004–1022.
- ✓ Pranzo M. , 2004. *Batch scheduling in a two-machine flow shop with limited buffer and sequence independent setup times and removal times.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH 153: 581–592.
- ✓ Qian B. , Wang L. , Wang D.-X., Wang W.-L., Wang X., 2007. *An effective hybrid de-based algorithm for multi-objective flow shop*

scheduling with limited buffers. COMPUTERS & OPERATION RESEARCH 39: 209–233.

- ✓ Quinlan J. R., 1986. *Introduction of decision trees.* MACHINE LEARNING 1: 80–106.
- ✓ Rahimi-Vahed A. R., Mirghorbani S. M., 2006. *A multi-objective particle swarm for a flow shop scheduling problem.* JOURNAL OF COMBINATORIAL OPTIMIZATION 13: 79–102.
- ✓ Rashidi E., Jahandar M., Zandieh M. , 2010. *An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY 49: 1129–1139.
- ✓ Rathinasamy S. , R R., 2009. *Sequencing and scheduling of nonuniform flow pattern in parallel hybrid flow shop*. THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY 49: 213–225.
- ✓ Ribas I., Leisten R., Framinan J. M., 2010. *Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective.* COMPUTERS & OPERATIONS RESEARCH 37: 1439–1454.
- ✓ Ruiz R., Stutzle T., 2007. *A simple and effective iterated greedy algorithm for permutation flowshop scheduling problem.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH 177: 2033–2049.
- ✓ Ruiz R., Serifoglu F. S., Urlings T. , 2008. *Modeling realistic hybrid flexible flowshop scheduling problems.* COMPUTERS & OPERATIONS RESEARCH 35: 1151 – 1175.
- ✓ Ruiz R., Vazquez Rodriguez J. A., 2010. *The hybrid flow shop scheduling problem.* EUROPEAN JOURNAL OF OPERATIONAL RESEARCH 205: 1–18.
- ✓ Sawik T., 2000. *Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers.* MATHEMATICAL AND COMPUTER MODELLING 31(13): 39-52.
- ✓ Sawik T., 2002. *An Exact Approach for Batch Scheduling in Flexible Flow Lines with Limited Intermediate Buffers.* MATHEMATICAL AND COMPUTER MODELLING 36: 461-471.

- ✓ Serifoglu F., Ulusoy G., 2004. *Multiprocessor task scheduling in multistage hybrid flow-shops: A genetic algorithm approach.* JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY 55: 504–512.
- ✓ Sevastianov S.V., 2002. *Geometrical heuristics for multiprocessor flowshop scheduling with uniform machines at each stage.* JOURNAL OF SCHEDULING 5: 205-255.
- ✓ Sha L., Abdelzaher D., Arzen K. E., Cervin A., Baker T., Burns A., Buttazzo G., Caccamo M., Lehoczky J., Mok A. K. , 2004. *Real-time scheduling: a Historical Perspective.* REAL-TIME SYSTEMS 28: 101-155, KLUVER ACADEMIC PUBLISHERS.
- ✓ Strocchi E., 2006. *Rappresentazione schematica della scomposizione in sottoproblemi utilizzata nel branch and bound.* E. STROCCHI'S WEBSITE (http://www.unibo.it/docenti/enrico.strocchi).
- ✓ Tang L., Liu W., Liu J., 2005. *A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment.* JOURNAL OF INTELLIGENT MANUFACTURING 16: 361–370.
- ✓ Tseng C.-T., Liao C.-J., 2008*. A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks.* INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH 46: 4655–4670.
- ✓ Vakharia A. J., Moily J. P., Huang Y., 2000. *Evaluating virtual cells and multistage flow shops: an analytical approach.* THE INTERNATIONAL JOURNAL OF FLEXIBLE MANUFACTURING SYSTEMS 11: 291–314.
- ✓ Voß S., Witt A., 2005. *Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: a real-world application.* INTERNATIONAL JOURNAL OF PRODUCTION ECONOMICS 105: 445–458.
- ✓ Wang H.-M., Chu F.-D., Wu F.-C., 2010. *A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY, 28 JULY (DOI 10.1007/S00170-010-2868-Z).

- ✓ Wang L., Zhang L., Zheng D.-Z., 2005. *A class of hypothesis-test-based genetic algorithms for flow shop scheduling with stochastic processing time.* THE INTERNATIONAL JOURNAL OF ADVANCED MANUFACTURING TECHNOLOGY 25: 1157–1163.
- ✓ Wittrock R. J., 1985. *Scheduling algorithms for flexible flow lines.* IBM JOURNAL OF RESEARCH AND DEVELOPMENT 29: 401-412.
- ✓ Xuan H., Tang L., 2007. *Scheduling a hybrid flow shop with batch production at the last stage.* COMPUTERS & OPERATIONS RESEARCH 34: 2718-2733.
- ✓ Ying K.-C., Lin S.-W., 2006. *Multiprocessor task scheduling multistage hybrid flow-shops: an ant colony system approach.* INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH 44: 3161–3177.
- ✓ Zandieh M., Fatemi Ghomi S. M. T., Moattar Husseini S. M., 2006. *An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times.* APPLIED MATHEMATICS & COMPUTATION 180: 111–127.
- ✓ Zandieh M. , Mozaffari E., Gholami M. , 2009. *A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems.* JOURNAL OF INTELLIGENT MANUFACTURING 21: 731–743.
- ✓ Zitzler E., Laumanns M., Thiele L., 2001. *SPEA-II: Improving the strength pareto evolutionary algorithm.* COMPUTER ENGINEERING AND NETWORKS LABORATORY.