27. Scheduling and Production & Control: MA

Pablo Moscato, Alexandre Mendes and Carlos Cotta

27.1 Introduction

This chapter addresses three main problems of the production planning area: Single Machine Scheduling (SMS), Parallel Machine Scheduling (PMS) and Flowshop Scheduling (FS). Many situations in production environments found in industries around the world can be modelled as one or a set of them (Baker 1974). It is easy to find dozens of problem variants in the literature due to different production constraints and objective functions. Due to the overwhelming number of combinations, we focused our discussion on only one of each problem class. They are:

- SMS with sequence-dependent setup times and with the objective of minimizing of the total tardiness.
- PMS with sequence-dependent setup times and minimization of the makespan.
- FS with families of jobs, sequence-dependent setup times and minimization of the makespan.

The three problems are NP-hard. That means finding that under the current conjecture that the computational complexity classes P and NP are not equal, it is widely conjectured that the optimal solution of an arbitrary instance of these problems cannot be found in polynomial time (Lawler et al. 1993). They also share sequence-dependent setup times, which are common in industrial environments and dramatically increase the complexity of the problem. These times come from the fact that the machinery usually requires some kind of maintenance (cleaning, alignment, tooling change, etc.) before starting the production of a given job, or sets of jobs. The objective functions treated are the *total tardiness* and the *makespan*.

The total tardiness obliges the scheduling to respect production due dates, defined by the clients that make the orders. They are utilized when the industry wants to deliver the customers' orders on time, or with the least delay. Moreover, no penalty is applied if products are completed before their due dates. In other words, no after-production inventory costs are considered.

The makespan measures the total time of production, marked by the time span between the production start and the time when the last job leaves the shop floor. When no setup times are considered, the makespan is constant, but with setup times, the minimization of this value becomes a challenge. Other important considerations of the problems addressed are as follows:

- A set on independent, single-operation, non-preemptive jobs is available with *ready-time*[1] zero.
- The machines are continuously available and have no limitation of resources.
- Tooling availability is not a production-limitation factor.
- No machine breakdowns or other contingencies are considered.
- All parameters defining the production environment – processing times, setup times and due dates – are previously known.

27.2 The single machine scheduling problem

The Single Machine Scheduling is one of the first studied class of problems in the scheduling area (Graham et al. 1979; Graves 1981). There are many different types of SMS problems (generally due to different input data and objective functions). One of the simplest to state, but not easy to solve at all, is the problem of sequencing n jobs, given its processing times and due dates (distinct for each job), and with the objective function being to minimize the total tardiness. Tardiness is a *regular* performance measure, which means that an optimal schedule cannot have idle times between jobs. Moreover, the tardiness increases only if at least one of the completion times[2] in the schedule also increases. Under these circumstances, a valid solution of the problem can be defined as a simple permutation of the jobs. Using the permutation space representation, the solution space has $n!$ configurations, all of them representing valid solutions of the problem.

The SMS problem addressed can be extended by the inclusion of precedence constraints for the jobs, ready-times, resources limitations, etc. From an objective function's point-of-view, we may want to minimize the *makespan*, the *total tardiness*, the *mean tardiness*, the number of *tardy jobs*, or even a combination of these objectives, which would characterize a *multi-criteria*[3] problem.

It is easy to see the large variety of problems that we may face in practice (Gen and Cheng 1997). Even looking at very complex industrial manufacturing systems, it is not hard to find situations in which a simple SMS should be solved (Ow and Morton 1989). In multiple-operation processes, a single part usually goes through several machines, where each machine performs a specific job at the part. Depending on the processes involved, there might be bottlenecks in this sequence, which can be modelled as SMS problems. Intricate jobs, performed by dedicated machines fit in that category. These machines are usually very expensive and probably there will be only one of them in the production environment, being used continuously. This situation represents a bottleneck and the schedule determina-

[1] Ready-time is the time that the job becomes ready to start being processed.

[2] Completion time is the time when a given job finishes its processing.

[3] In multi-criteria problems, two or more performance criteria, sometimes confronting ones, shall be optimised. These problems are harder to solve, mainly because of the difficulty to compare the solutions' quality. In such cases, Pareto's optimality criteria should be employed.

tion for this machine will affect the production as a whole. Next, it is shown the description of the SMS problem addressed in this chapter:

Input: Let n be the number of jobs to be processed in one machine. Let $P = \{p_0, p_2, ..., p_n\}$ be the list of processing times, $D = \{d_1, d_2, ..., d_n\}$ be the list of due dates for each job and $S_0 = \{s_{01}, s_{02}, ..., s_{0n}\}$ be the list of initial setup times. Let $\{S_{i,j}\}$ be a matrix of setup times, where $s_{i,j}$ is the time required to set up job j after the machine has just finished processing job i.

Task: Find a permutation that minimizes the total tardiness of the schedule. Suppose that the jobs are scheduled in the order $\{\pi(1), \pi(2), ..., \pi(n)\}$. Then the total tardiness can be calculated using the Eq. 27.1.

$$Total_{tardiness} = \sum_{k=1}^{n} \max\left[0, c_{\pi(k)} - d_{\pi(k)}\right] \tag{27.1}$$

Where $c_{\pi(k)}$ represents the *completion time* of job $\pi(k)$. The $c_{\pi(k)}$ values can be calculated using the Eq. 27.2.

$$c_{\pi(k)} = s_{0\pi(1)} + p_{\pi(1)} + \sum_{i=2}^{k}\left[s_{\pi(i-1),\pi(i)} + p_{\pi(i)}\right] \tag{27.2}$$

It is well known that the problem of sequencing jobs in one machine without setup times is already NP-hard (Du and Leung 1990). Despite its application to real world settings, the SMS problem addressed in this chapter has received little attention in the scheduling literature. In (Ragatz 1993) a *branch-and-bound* (B&B) method is proposed, but only small instances could be solved to optimality. The papers of (Raman et al. 1989) and (Lee et al. 1997) use dispatch rules based on the calculation of a priority index to build an approximate schedule, which is then improved by the application of a local search procedure. The ATCS heuristic presented in (Lee et al. 1997) had an impressive performance for this problem, considering its simplicity. Three papers using metaheuristics have been proposed so far. In (Rubin and Ragatz 1995) a new crossover operator was developed and a *genetic algorithm* (GA) was applied to a set of test problems. The results obtained by the GA approach were compared with the ones from a B&B and with a *multiple start* (MS) algorithm and they concluded that MS outperformed the B&B and the GA in many instances, considering running time and quality of solutions as performance measures. Of course, the instances in which MS outperformed B&B were the ones where the exact method did not find an optimal solution before a limit on the number of nodes was reached. The B&B was truncated in such cases, returning sub-optimal schedules. Given these results, (Tan and Narasimhan 1997) chose the MS technique as a baseline benchmark for conducting comparisons with the *simulated annealing* (SA) approach they proposed. Their conclusion was that SA outperformed MS in all but three instances, with percentage improvements not greater than 6%. More recently, (França et al. 2001) proposed a new *memetic algorithm* (MA) that outperformed the previous approaches. The results presented in this chapter are an improvement over previously reported results. The BOX cross-

over, together with the multi-population approach (both equal to those described in Chap. 18) produced a big performance leap.

27.2.1 The test instances

The test set used for the experimental analysis consists of instances that vary from 17 to 100 jobs. This set was constructed from *Asymmetric Travelling Salesman Problem* (ATSP) instances which optimal solutions are available in the literature. All ATSP instances used for this purpose in this work, as well as their optimal solutions, are available at the TSPLIB website[4].

It is clear that the ATSP and the SMS problems share similarities and they have been highlighted for decades. The most important relates the SMS's *setup times* with the ATSP's *distance matrix*. This correspondence suggests a transformation procedure to create an "interesting set" of SMS instances starting from an arbitrary ATSP instance with known optimal solution. We propose a three-step procedure to generate such SMS instances. Each SMS instance is composed of a matrix of setup times and three lists of n integers. In the transformation, the setup times matrix will be equal or be a multiple of the ATSP's distance matrix. Then, a simple procedure generates the three lists required to complete the SMS instance. Suppose that the permutation $\{\pi(1), \pi(2), ..., \pi(n)\}$ represents the sequence of cities in the optimal tour for the ATSP instance.

Step 1 - Generation of the S_0 list: Let $[\pi(k), \pi(k+1)]$ be the pair of adjacent nodes of the ATSP optimal tour with the largest distance between cities (named *dist*). Let $s_{0,\pi(k+1)}$ be equal to the cost of this arc, $dist_{\pi(k),\pi(k+1)}$. Let all other s_{0j}, $j \neq \pi(k+1)$ be greater than $s_{0,\pi(k+1)}$. Moreover, the initial job of the optimal SMS solution will be $\pi(k+1)$.
Step 2 - Generation of the P list: Initially, re-enumerate the nodes so that $\pi(1) \Rightarrow \pi(k+1)$; $\pi(2) \Rightarrow \pi(k+2)$; ...; $\pi(n) \Rightarrow \pi(k)$. Then, construct the list $\{p_{\pi(1)}, p_{\pi(2)}, ..., p_{\pi(n)}\}$ in such a way that $p_{\pi(1)} < p_{\pi(2)} < ... < p_{\pi(n)}$.
Step 3 - Generation of the D list: Construct the due dates in such a way that $d_{\pi(1)} < d_{\pi(2)} < ... < d_{\pi(n)}$ and $d_{\pi(i)} \in [c_{\pi(i)} - p_{\pi(i)}, c_{\pi(i)}] \; \forall \; \pi(i)$: $i = 1, ..., n$, where $c_{\pi(i)}$ is the completion time of job $\pi(i)$ following the ATSP optimal sequence.

It is worth emphasizing that we are not stating that the transformation procedure described above preserves the ATSP optimal tour permutation as the SMS optimal sequence. Consequently, it is an open problem to find the optimal tardiness value of the resulting SMS instances. Nevertheless, in all tests we have executed (using the TSPLIB instances as the basis of the transformation) we have never reached a better tardiness than the one obtained with the jobs in the ATSP optimal permutation. As a consequence, we will assume that such values can be considered at least as very high-quality upper bounds.

[4] http://www.crpc.rice.edu/softlib/tsplib/

We will now describe the intuition behind the suggested transformation. The direct use of the distance matrix as the setup times between jobs is pretty clear. Suppose that all processing times are zero in the SMS problem. In this case, the SMS problem is basically a generalization of the ATSP on which we have the extra constraint of due dates to visit each city.

The idea of starting the sequence with the setup corresponding to the largest distance belonging to the optimal ATSP tour aims to avoid that a change in the initial job produces a reduction in the sum of the completion times, what could make the optimal sequence be lost. Another important point is the requirement that $dist_{\pi(k+1),\pi(k)} < dist_{\pi(k),\pi(k+1)}$. However, these rules are not enough. Let us show an example where the transformation does not preserve optimality (see Fig. 27.1).

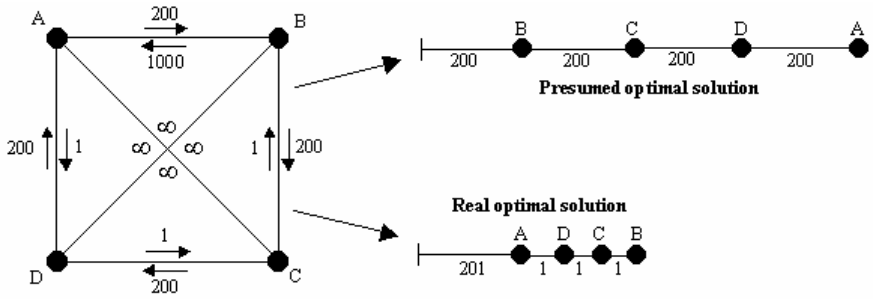

Fig. 27.1. An example where the optimal solution of the ATSP instance is not the optimal permutation of the SMS instance.

In Fig. 27.1, on the left side there is an ATSP instance composed of four cities. As the internal arcs are valued "infinite", the only two tours that avoid using one of these edges are [B C D A] – with a total length of 800 – and [A D C B] – with a total lenght 1003. Therefore, consider the optimal tour [B C D A]. For the sake of simplicity, ignore the processing times and due dates of the corresponding SMS problem as we are interested only in the effect caused by the setup times. On the upper-right portion of the Fig. 27.1, the presumed optimal sequence is shown, with a makespan of 800 – equal to the optimal ATSP tour length. Consider the situation where the initial arc has a very high value in the opposite direction. In this case, according to the transformation procedure, it could be replaced by a value greater than $s_{0,\pi(k+1)}$ but smaller than the original. In the example, it was substituted by 201, what is much smaller that 1000. The consequence is that the real optimum makespan becomes lower than the presumed one and the transformation fails to preserve optimality. If in Step 1, the restriction $dist_{\pi(k+1),\pi(k)} < dist_{\pi(k),\pi(k+1)}$ is not observed, the transformation might not be valid. Thus, a necessary condition is that the initial arc must always have a larger value when it is considered in the direction of the presumed optimal solution than when it is considered in the opposite direction.

The steps 2 and 3 have the same function. When we order the processing times and the due dates following an increasing sequence, we avoid that any change in a job position leads to a reduction in total tardiness. The way that the *P* and *D* lists

are generated produces instances that are solvable to optimality by the *Shortest Processing Time* (SPT) and by the *Earliest Due Date* (EDD) heuristics. Methods that use such procedures, or their variants, will solve the instances instantly, consequently producing biased results. Nevertheless, in order to evaluate the memetic algorithm, which does not employ any such heuristics in its operators, the instances are absolutely adequate.

27.2.2 The memetic algorithm approach

The MA utilized in the SMS problem is very similar to that presented for the Gate Matrix Layout problem (see Chap. 18). Therefore, we refer the reader to that chapter and we will focus only on a few minor differences between them.

The representation chosen is quite intuitive, with a solution being represented as a chromosome with the alleles assuming different integer values in the [1, n] interval. In other words, a solution is a permutation of n integers.

Two crossover operators were tested: BOX and OX. In the first one, after choosing two parents, several fragments of the chromosome from one of them are randomly selected and copied into the offspring. In a second phase, the offspring's empty positions are sequentially filled with the alleles present in the chromosome of the other parent, from left to right. The procedure tends to perpetuate the relative order of the jobs and has a better overall performance over the original OX crossover (Goldberg 1989), which selects a single block of one of the parents to be copied into the offspring. In this implementation, the parent that copies its block(s) into the offspring is the leader of the cluster, and the supporter will complete the empty positions (see Fig. 27.2).

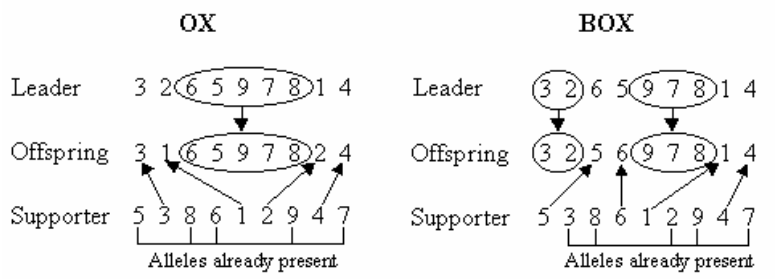

Fig. 27.2. Crossover operators for the SMS problem – OX and BOX

The number of individuals created in every generation is very high – two times the number of individuals in the population. This value is consequence of the acceptance policy for the offspring, which is very restrictive. Therefore in order to balance the large number of discarded individuals, we must create many of them every generation.

In this implementation, we employed the *all-pairs* and the *insertion* neighbourhoods, just like in the VLSI problem. Due to the complexity of each individual evaluation, a neighbourhood reduction based on the setup times' values had to be

developed. It was observed that most good schedules have a common characteristic: the setup times between jobs in these solutions are very small. This is reasonable, since schedules with small setup times between jobs will be less lengthy, generating fewer delays. Next, we show two diagrams illustrating how the schedule is modified if jobs *i* and *j* swap their positions (see Fig. 27.3) and if job *i* is inserted immediately before job *j* (see Fig. 27.4).

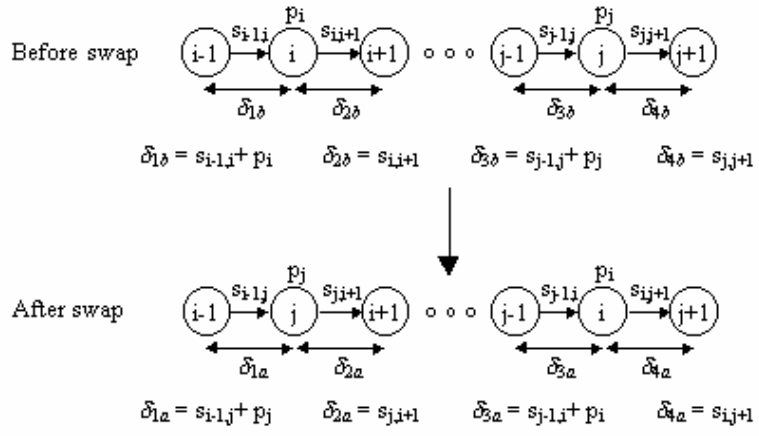

Fig. 27.3. Diagram of a swap move in an all-pairs neighbourhood and its implications on the schedule

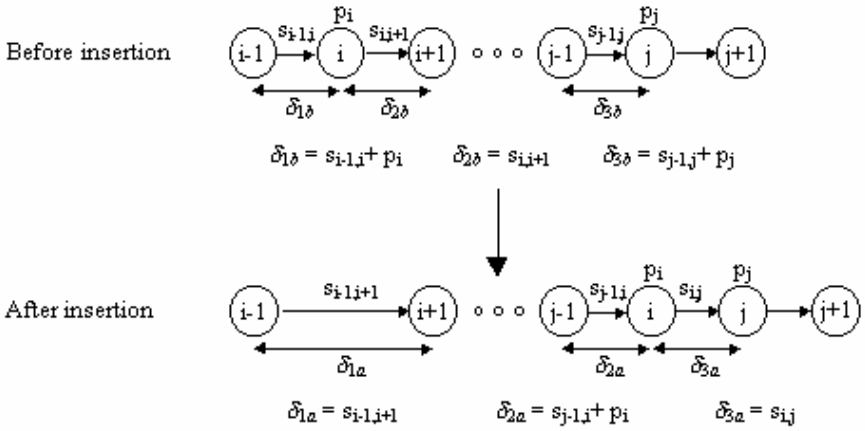

Fig. 27.4. Diagram of a move in an insertion neighborhood and its implications on the schedule

Based on the δ variations, it is possible to create rules to determine if a given move should be evaluated or not. Among the several possibilities of combinations, the best results came, for the all-pairs neighbourhood, if we evaluate the swap move only if '*at least one* δ *value improves in both modified regions*'. For the in-

sertion neighbourhood, the best policy is to test an insertion move only '*if the sum of the δ variables in at least one modified region improves*'. In mathematical form, these rules can be written as logical clauses:

- $[(\delta_{1a} < \delta_{1b}) \vee (\delta_{2a} < \delta_{2b})] \wedge [(\delta_{3a} < \delta_{3b}) \vee (\delta_{4a} < \delta_{4b})]$, for the all-pairs neighbourhood.
- $(\delta_{1a} < \delta_{1b} + \delta_{2b}) \vee (\delta_{2a} + \delta_{3a} < \delta_{3b})$, for the insertion neighbourhood.

Since each individual evaluation requires a considerable computational effort[5] reductions like these save a lot of time. In fact, although the reduction policies are very strict, reducing the neighbourhoods to less than 10% of their original size, they maintain the search focused on promising moves. More information on the several reduction policies tested, as well as performance comparisons, can be found in (França et al. 2001).

27.2.3 The SMS computational results

As said before, the SMS instances use the ATSP distance matrices. The only flexible parameters are the processing times and the due dates. By combining these two components of the input, we classified the instances in four groups. The parameters are generated according to the rules:

Processing Times:

LOW:	$p_k \in [0, \frac{1}{4}.\max(s_{ij})]$	$\forall\, k: k = 1, ..., n$
HIGH:	$p_k \in [0, 2.\max(s_{ij})]$	$\forall\, k: k = 1, ..., n$

Due dates:

HARD:	$d_k = c_k$	$\forall\, k: k = 1, ..., n$
SOFT:	$d_k \in [c_k - p_k, c_k]$	$\forall\, k: k = 1, ..., n$

The *LOW* rule makes the setup times a more critical aspect in the problem, emphasizing its "ATSP-like character". In these instances, the processing times are small, compared to the setup times. The *HIGH* policy, on the other hand, makes the processing times be more relevant in the schedule determination. Their values can be up to two times larger than the maximum setup time.

The *HARD* policy generates instances in which the total tardiness of the schedule that corresponds to the optimal ATSP solution equals to zero. This value in guaranteed because the due dates are placed over the jobs' completion times, when the production follows the optimal ATSP tour. Nevertheless, it is a quite difficult problem, since only one – or perhaps a few – of the $n!$ solutions has a zero tardiness. The *SOFT* instances, on the other hand, have greater-than-zero optimal

[5] Each individual evaluation is $O(n)$ and the complete neighbourhoods are $O(n^2)$.

total tardiness and are better suited for making relative comparisons, where percentage deviations from optimal value are numerically necessary. Next, in Table 27.1, we show some characteristics of the original ATSP instances such as number of cities and distance between cities – minimum and maximum $dist_{ij}$.

Table 27.1. Number of cities (jobs), and distance-between-cities interval (setup times) of the ATSP instances

Instance name	n	Minimum $dist_{ij}$	Maximum $dist_{ij}$
br17	17	0	74
ftv33	34	7	332
ftv55	56	6	324
ftv70	71	5	348
kro124p	100	81	4,545

The instances tried to cover a wide range of number of jobs. Moreover, the resulting SMS setup times also cover a wide range. The Tables 27.2 and 27.3 show all the results obtained for the set of instances created. The name of the instance is in the first column and is divided in two parts. The initial part refers to the name of the original ATSP instance. Then, the two capital letters indicate if the instance is LOW or HIGH, and HARD or SOFT. The second column indicates the presumed optimal tardiness. Next to it, we have the average total tardiness found using the OX and the BOX recombination operators, considering 10 runs for each instance. Written in bold letter, subscript, there is the number of times that the optimal solution was found, out of 10 trials. Finally, in the columns labeled 'CPU time' it is shown the average CPU time.

Single population tests

The Table 27.2 shows the results for the single population memetic algorithm, using the OX and BOX crossovers. The maximum CPU time was fixed at four minutes, but several times the "presumed optimal" (i.e. the permutation corresponding to the optimum ATSP tour) was reached before this limit. The equipment utilized is a 366 MHz Pentium II Celeron with 128 MB RAM.

The results in Table 27.2 show that the memetic algorithm successfully solved most of the instances. The OX crossover had a worse overall performance, compared to the BOX, failing in many of the 100-job instances. Concerning this criterion, the finding of 167 and 180 optimal solutions – out of 200 possible – for the OX and BOX crossovers, respectively, can be considered a very reasonable rate of success. Based on the results, we can also conclude that the 100-job instances are in the threshold of the search capability of the algorithms, because the number of optimal solutions decreased considerably. For the *kro124pLS*, for example, the "presumed optimal" was found only once in twenty trials. Probably, for instances with more than 100 jobs, the method might fail to find any optimal solutions and more features will have to be included in the algorithm in order to increase its search power. Another important point is the CPU time required by each configuration. In general, the BOX needs less CPU time than the OX to find the optimal

solutions. This characteristic becomes more evident in the *ftv70* and *kro124p* sets of instances.

Table 27.2. Results for the single population memetic algorithm

Instance name	Presumed optimal tardiness	OX Average Tardiness	BOX average tardiness	OX CPU time (sec.)	BOX CPU time (sec.)
br17LH	0	0_{10}	0_{10}	0.1	0.1
br17LS	52	52_{10}	52_{10}	0.1	0.1
br17HH	0	0_{10}	0_{10}	0.1	0.1
br17HS	547	547_{10}	547_{10}	0.1	0.1
ftv33LH	0	0_{10}	0_{10}	2.1	2.1
ftv33LS	664	664_{10}	664_{10}	2.8	2.6
ftv33HH	0	0_{10}	0_{10}	1.3	1.3
ftv33HS	5,324	$5,324_{10}$	$5,324_{10}$	1.5	1.4
ftv55LH	0	0_{10}	0_{10}	17.4	13.8
ftv55LS	1,170	$1,170_{10}$	$1,170_{10}$	50.9	19.8
ftv55HH	0	0_{10}	0_{10}	6.9	8.0
ftv55HS	8,515	$8,515_{10}$	$8,515_{10}$	12.9	14.5
ftv70LH	0	918.6_{7}	0_{10}	148.5	46.0
ftv70LS	1,506	$1,557.4_{8}$	$1,506_{10}$	165.0	70.1
ftv70HH	0	0_{10}	0_{10}	41.8	23.6
ftv70HS	12,368	$12,368_{10}$	$12,368_{10}$	40.3	35.3
kro124pLH	0	$23,872.0_{2}$	$22,717.1_{3}$	229.5	205.5
kro124pLS	26,111	$74,636.3_{0}$	$68,618.8_{1}$	240.0	224.3
kro124pHH	0	$4,710.6_{7}$	0_{10}	157.3	107.1
kro124pHS	223,890	$238,114.2_{3}$	$231,997.5_{6}$	220.4	185.5
* Total optimal solutions found		***167**	***180**		

Multiple population tests

The multiple population memetic algorithm aims to validate the use of multiple populations for the SMS problem. Using the results presented in the Chap. 18 – Gate Matrix Layout Problem – as a guideline, the number of populations was fixed at four and the migration policy used was the so-called *1-Migrate*. The maximum CPU time remained at four minutes and the results are presented in Table 27.3.

The multiple population approach had a better performance than the single population one. The number of presumed optimal solutions found was greater – 170 and 189 against 167 and 180 in the single population version. With the use of the BOX crossover, we attained very strong results even for the 100-job instances, what makes us believe that the method could deal with larger problems. In this test, the relation between CPU time's requirements was the same. The BOX is much more efficient than the OX, reaching the optimal solutions much faster. Still regarding the CPU time, sometimes the multiple population approach is slower than the single population, especially for the smaller instances. This was already expected since the algorithm had to evolve four populations instead of only one.

The problem must have a sufficient size – and complexity – in order to take advantage of the *genetic drift* effect (see Chap. 18) and be noticeable from computer experiments.

Table 27.3. Results for the multiple population memetic algorithm

Instance name	Presumed optimal tardiness	OX average tardiness	BOX Average Tardiness	OX CPU time (sec.)	BOX CPU time (sec.)
br17LH	0	0_{10}	0_{10}	0.1	0.1
br17LS	52	52_{10}	52_{10}	0.1	0.1
br17HH	0	0_{10}	0_{10}	0.1	0.1
br17HS	547	547_{10}	547_{10}	0.1	0.1
ftv33LH	0	0_{10}	0_{10}	1.7	2.1
ftv33LS	664	664_{10}	664_{10}	3.4	2.6
ftv33HH	0	0_{10}	0_{10}	1.2	1.6
ftv33HS	5,324	$5{,}324_{10}$	$5{,}324_{10}$	1.6	2.0
ftv55LH	0	0_{10}	0_{10}	43.8	13.6
ftv55LS	1,170	$1{,}170_{10}$	$1{,}170_{10}$	40.1	22.5
ftv55HH	0	0_{10}	0_{10}	8.4	8.1
ftv55HS	8,515	$8{,}515_{10}$	$8{,}515_{10}$	12.8	8.7
ftv70LH	0	0_{10}	0_{10}	122.7	48.3
ftv70LS	1,506	$1{,}684.7_{8}$	$1{,}506_{10}$	172.9	53.6
ftv70HH	0	0_{10}	0_{10}	39.2	18.0
ftv70HS	12,368	$12{,}368_{10}$	$12{,}368_{10}$	50.8	35.1
kro124pLH	0	$48{,}536.5_{1}$	$26{,}175.9_{6}$	231.7	206.6
kro124pLS	26,111	$72{,}553.7_{0}$	$53{,}807.4_{4}$	240.0	227.7
kro124pHH	0	$3{,}337.5_{8}$	0_{10}	159.3	118.9
kro124pHS	223,890	$240{,}213.1_{3}$	$233{,}074.3_{9}$	228.1	141.3
* Total optimal solutions found		***170**	***189**		

27.3 The parallel machine scheduling problem

The second part of this chapter addresses the PMS problem. It consists of scheduling a given set of *n* jobs to *m* identical parallel machines with the objective of minimizing the *makespan*. As in the SMS problem, there are sequence-dependent setup times between jobs. The PMS can be considered a generalization of the SMS since it takes into account several machines, instead of only one. In fact, the solution found to deal with bottleneck situations like those described in Sect. 27.2 is usually the addition of more machines, transforming the SMS problem into a PMS one. Of course, sometimes this cannot be done, generally due to financial limitations. The resulting problem can be faced as two problems. The first is how to assign the jobs to the several machines and the second is how to schedule the jobs in each machine. The PMS with setup times is frequently found in real world settings where the setups are not negligible, especially in the paper, chemical and textile industries.

Previous PMS-related works are dated back to 1984. In that year, Dearing and Henderson (1984) developed an integer linear programming model for loom assignment in a textile weaving operation. They reported results found through the rounding of the solutions obtained by the linear relaxation of the integer model. In 1987, Sumichrast and Baker (1987) proposed a heuristic method based on the solution of a series of 0-1 integer subproblems, improving the results obtained in (Dearing and Henderson 1984). These two articles deal with a slightly different problem because they assume that a job can be split among several machines. The case being addressed in this chapter is combinatorially more complex. To the authors' knowledge, very few papers have reported computational results for this problem. The work of Frederickson et al. (1978) present approximate algorithms derived from an equivalence between the PMS and the *Travelling Salesman Problem* (TSP). França et al. (1996) used a *tabu search*-based (TS) heuristic in connection with a powerful neighborhood scheme that employs the concept of local and global neighbors. Later, Mendes et al. (2002) proposed a MA approach for the problem, comparing the results with the ones found in (França et al. 1996).

The PMS problem is a difficult combinatorial problem proved to be NP-hard in a strong sense because it is equivalent to the TSP when the number of processors equals one (Baker 1974). Next, it is shown the description of the PMS problem.

Input: Let n be the number of jobs to be processed in m identical machines. Let $P = \{p_1, p_2, ..., p_n\}$ be the list of the jobs' processing times, and $S_0 = \{s_{01}, s_{02}, ..., s_{0n}\}$ be the list of initial setup times. Let $\{S_{ij}\}$ be a matrix of setup times, where s_{ij} is the time required to set up job j after the machine has just finished processing job i.

Task: Assign the jobs to the machines and find the permutation in each machine that minimizes the production makespan. In order to calculate the makespan, initially consider that $\pi(k,l)$ represents the k-th job of machine l. The total production time in machine l, represented by Γ_l, can be calculated by the Eq. 27.3.

$$\Gamma_l = s_{0\pi(1,l)} + p_{\pi(1,l)} + \sum_{i=2}^{n_l} \left[s_{\pi(i-1,l),\pi(i,l)} + p_{\pi(i,l)} \right] \tag{27.3}$$

Where n_l is the number of jobs processed by machine l. Thus, the makespan is the maximum production time among all machines, being represented by the Eq. 27.4.

$$Makespan = \max_i \left[\Gamma_i \right] \tag{27.4}$$

27.3.1 The test instances

The instances used in the computational experiments were randomly generated. The number of jobs was fixed at 20, 40, 60 and 80, and the number of machines at 2, 4, 6 and 8. Processing times were generated following a discrete uniform distribution DU(1, 100). Setup times were divided into two categories: small setup times – with values in the interval [1, 10] – and large setup times – with values in

the interval [1, 100]. The setup times were also generated according to two possibilities: structured and non-structured. The structured setup times follow the triangular inequality, that is, $s_{ij} \leq s_{ik} + s_{kj}$, $\forall\ i, j, k$; $k \neq i, j$. The non-structured setup times do not follow the triangular inequality. We considered 10 replications for each combination of number of jobs and machines.

27.3.2 The memetic algorithm approach

The MA utilized for the PMS problem is very similar to the GMLP (see Chap. 18, this book) and the SMS. The differences are concentrated on the crossover type – the BOX was replaced by the OX – and on the use of a single population, instead of several ones. Now, let us begin by addressing the individual representation for this problem.

The representation chosen for the PMS is a chromosome with its alleles assuming different integer values in the [1, n] interval. In order to include information about the m machines, m-1 *cut-points* (represented by a '*' symbol) were introduced to define the jobs assignment to each machine. For instance, a chromosome [1 4 6 * 3 7 2 10 * 8 9 5] is a possible solution for a problem with 10 jobs and 3 machines. The cut-points are in positions 4 and 9. Thus, machine 1 executes operations [1 4 6], in this order; machine 2 executes operations [3 7 2 10] and machine 3 performs operations [8 9 5]. As the machines are identical, no distinction must be made between the cut-points.

As said before, the crossover operator implemented is the Order Crossover (OX). Because of the cut-points' role and their effect in the schedule, the OX in this case behaves similarly to the BOX. The reason is that any cut-point position change influences the job/machine assignment of the entire sequence, resembling the kind of perturbation that would be caused by the BOX. The difference between the OX implemented and the originally introduced by (Goldberg 1989), is that the offspring is filled from the beginning of the sequence and not after the piece copied from the first parent. The Fig. 27.5 shows a diagram of how the crossover works.

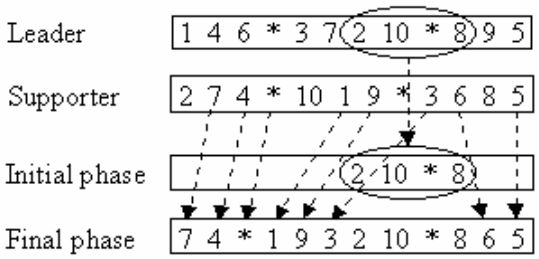

Fig. 27.5. Diagram of the OX crossover used for the PMS problem

In the Fig. 27.5, the initial fragment inherited from the *leader* parent consists of the alleles [2 10 * 8], and they are copied to the same positions in the offspring.

The offspring's empty positions were then filled according to the order that the alleles appear in the chromosome of the *supporter* parent. Repeated alleles are skipped as well as cut-points, if there are already m-1 cut-points present in the offspring.

The local search utilized is based on the *all-pairs* swap and the *insertion* neighbourhoods and is applied to every new individual. No reduction scheme was employed for this problem, since the largest instance consisted of 80 jobs and 8 machines, which is still a computationally tractable size.

27.3.3 The PMS computational results

The previous best results in the literature on the PMS problem are presented in (França et al. 1996) and (Mendes et al. 2002). In (França et al. 1996) it was also introduced a set of instances for which optimal solutions or high quality lower bounds were calculated. In Table 27.4, we compare the MA against these values and also against upper bounds obtained by an intensively executed *tabu search* (TS) algorithm, named *long tabu*. The long tabu usually required several hours of execution, but at the end, high quality *upper bounds* were found.

The percentage deviation from the lower bound/optimal solution is the performance measure for problems with 2 and 4 machines. The exception is the 4-machine problem with structured instances and $s_{ij} \in [1, 10]$. For this problem set and also for all structured problems with 6 and 8 machines, the deviation is related to the long tabu upper bounds.

A variant of the long tabu method is also presented in (França et al. 1996). It is a faster, less complex TS named *fast tabu*. Although less powerful, the fast tabu method is very efficient in finding good schedules for the PMS problem. Since the computational time required by this algorithm was much lower compared to the long tabu, we concluded it would be a fair opponent for the MA. Thus, the CPU times utilized by the MA are the same of the fast tabu, which vary from a few seconds up to five minutes, depending on the instance size.

Table 27.4. Results for the PMS instances

	Non-structured problems				Structured problems			
	$s_{ij} \in [1, 10]$		$s_{ij} \in [1, 100]$		$s_{ij} \in [1, 10]$		$s_{ij} \in [1, 100]$	
n / m	TS	MA	TS	MA	TS	MA	TS	MA
20 / 2	1.6	1.8	2.1	3.4	0.5	0.5	0.6	0.3
40 / 2	3.8	3.0	4.6	4.4	0.9	0.6	0.7	0.3
60 / 2	5.1	3.9	5.2	5.2	0.9	0.5	0.5	0.3
80 / 2	5.4	4.3	5.1	4.7	1.1	0.5	0.5	0.2
20 / 4	5.3	4.4	6.0	6.6	0.3	0.0	1.9	1.7
40 / 4	6.5	8.4	6.8	8.9	0.2	-0.1	1.6	1.3
60 / 4	7.0	9.1	7.2	9.9	0.3	-0.1	1.1	0.9
80 / 4	6.7	8.6	6.9	9.5	0.3	-0.1	0.8	0.8
20 / 6	0.8	1.3	1.6	1.1	0.7	-0.2	0.4	-0.8
40 / 6	1.0	4.1	0.8	3.8	0.2	0.3	0.2	0.1

60 / 6	0.4	4.5	0.7	5.0	0.2	0.2	0.2	0.1
80 / 6	0.7	4.2	0.3	4.5	0.2	0.1	0.3	0.3
20 / 8	1.4	1.7	1.0	0.6	1.0	-0.2	1.4	-0.7
40 / 8	1.0	6.6	1.1	4.1	0.2	0.2	0.7	0.2
60 / 8	0.7	5.3	0.8	5.2	0.1	0.3	0.3	0.4
80 / 8	0.4	4.8	0.9	5.5	0.1	0.2	0.3	0.5
Average	3.0	4.7	3.3	5.1	0.5	0.2	0.7	0.4

The MA was programmed in Java JDK 2.0 and run using a 366 MHz Pentium II Celeron. The TS were programmed in C and executed in a Sun Sparc 10 Workstation. The Java JDK 2.0 and the C compiler are very similar in terms of speed, with some advantage for the C compiler. On the other hand, the Sun Sparc 10 is a little slower than the Pentium II Celeron. For this reason we believe the speeds of both systems are somewhat equivalent, although this conclusion might not be accurate.

The Table 27.4 shows a comparison between the fast tabu and the MA approaches for the PMS problem. Looking at the averages row, it is clear that the methods had very different behaviours considering the instances structures. The non-structured instances were easier for the TS than for the MA. The opposite occurred for the non-structured ones. The deviations are very small in terms of percentage points, except for a few configurations where some figures are close to 10%. The negative figures in some of the MA columns mean that the makespans found were better than the upper bounds provided by the long tabu strategy.

There is a clear degradation in the MA as the number of machines increases, while the TS maintains an average performance, independently of the problem size. There is a strong probability that this was due to the local search employed in the MA. The simple *all-pairs* and *insertion* neighborhoods could be better suited for this problem if they utilized information about the cut-points. In fact, both local search and crossover operators are dealing with the PMS the same way they would deal with a SMS problem; all alleles are being treated equally. There is no distinction between jobs and cut-points from the genetic operators' point-of-view. More intelligent operators should use information of the problem's structure and the cut-points presence in the chromosome. For instance, the separation of the *job-to-machine* assignment and the *intra-machine* scheduling in the local search seems to be a reasonable starting point. A neighborhood reduction similar to that employed in the SMS should also be a good choice. Nevertheless, despite the relative lack of problem-driven properties, the general-use crossover and local search operators performed quite well against the well-tailored fast tabu. That increases our belief that the MA's refined structure is playing an important role in the algorithm's performance. Furthermore, as there is plenty of space for improvement in the operators, we believe that future contributions in this issue will probably make the MA surpass the TS in most instances types.

27.4 The flowshop scheduling problem

The last part of this chapter addresses a *flowshop* problem (FS), which main characteristic is to group the jobs in families. This is a quite common real-manufacturing characteristic since manufacturers want to take advantage of *group technology* (GT) environment (Schaller et al. 2000). In the GT scheduling problem, a part family is composed of several parts (in this work represented by the jobs) that have similar requirements in terms of tooling, setup costs and operations sequences. Usually, the families are assigned to a manufacturing cell based on operation sequences so that materials flow and scheduling are simplified. This process may result in a situation where each family is processed by a certain set of machines, and all jobs (parts) are processed following the same technological order.

In this production environment, the manufacturing cells resemble the traditional flowshops except for the existence of multiple part families. Since the jobs in the same family share similar tooling and setup requirements, usually a negligible or minor setup is needed to change from one part to another and thus can be included in the processing times of the jobs. However, a major sequence-dependent setup is needed to change the processing environment between two part families.

The FS with families of jobs is a difficult combinatorial problem proved to be NP-hard. When the number of jobs in each family equals one, the problem becomes a traditional FS problem with sequence-dependent setup times. This problem is proved to be NP-hard when the number of machines is greater than one (Gupta and Darrow 1986). Next, it is shown the description of the FS problem being addressed in this section.

Input: Let n be the number of jobs and m be the number of machines. All the jobs are processed following the same technological order, creating the flowshop structure. Let f be the number of families. Consider also a setup time to change the production from one family to another. Let $\{S_{i,j}^{l}\}$ represent these setup times, where $s_{i,j}^{l}$ is the setup time of family j after family i was processed, in machine l. Finally, let $\{P_{i,j}\}$ be a matrix of processing times, where $p_{i,j}$ is the processing time of job i in machine j.

Task: Find the permutation of the families, and of the jobs within these families, which minimizes the production *makespan*. Calculating the makespan for this problem is not an easy task. Let us initially suppose that the families are scheduled in the order $\{\pi(1), \pi(2), ..., \pi(f)\}$ and the order of the jobs within family f is given by the sequence $\{\sigma_f(1), \sigma_f(2), ..., \sigma_f(n_f)\}$, where n_f is the number of jobs in family f. Moreover, let $tt_{\sigma_f(i)}^{m}$ be the total processing time within family f, until job $\sigma_f(i)$, in machine m; i.e., the time span from when the machine finished its setup and is ready to process the first job of family f until the job $\sigma_f(i)$ is finished. This value can be calculated as:

$$tt_{\sigma_f(i)}^{m} = it_{\sigma_f(i)}^{m} + \sum_{z=1}^{i} p_{\sigma_f(z),m} \tag{27.5}$$

Where $it^{m}_{\sigma_f(i)}$ is the *idle time*[6] within family *f* accumulated until job $\sigma_f(i)$. In the first machine, no idle times are allowed, and the production flows without any interruption. That simplifies the equation, making it become the sum of the job's processing times within family *f*. But in the other machines, idle times might occur, depending on the schedule. The completion time of the *i*-th job of the *f*-th family in machine *m*, represented by $c^{m}_{\sigma_{\pi(f)}(i)}$ can thus be calculated as:

$$\underbrace{c^{m}_{\sigma_{\pi(f)}(i)} = \sum_{z=1}^{f-1}\left[tt^{m}_{\sigma_{\pi(z)}(n_{\pi(z)})} + s^{m}_{\pi(z),\pi(z+1)}\right]}_{\text{total time before family } \pi(f)} + \underbrace{tt^{m}_{\sigma_{\pi(f)}(i)}}_{\text{total time within family } \pi(f)} \tag{27.6}$$

The first part of the Eq. 27.6 calculates the total processing time before the *f*-th family, taking into account all the setup times, processing times and idle times before it. The second part calculates the total processing time within family $\pi(f)$ (processing times + idle times) until job $\sigma_{\pi(f)}(i)$. Now let us explicit the idle times calculation. Idle times occur always when a machine finishes processing a certain job, or completes the family setup, and the next job is still being processed in the previous machine. That creates a gap in the schedule, forcing the machine to wait until the next job becomes available. The idle time within family $\pi(f)$, accumulated until job $\sigma_{\pi(f)}(i)$ can be calculated as:

$$it^{m}_{\sigma_{\pi(f)}(i)} = \underbrace{\max\left(0, c^{m-1}_{\sigma_{\pi(f)}(1)} - c^{m}_{\sigma_{\pi(f-1)}(n_{\pi(f-1)})} + s^{m}_{\pi(f-1),\pi(f)}\right)}_{\text{idle time just before the first job of the } f\text{-th family}} + \tag{27.7}$$

$$+ \underbrace{\sum_{z=1}^{i}\max\left(0, +c^{m-1}_{\sigma_{\pi(f)}(z)} - c^{m}_{\sigma_{\pi(f)}(z-1)}\right)}_{\text{idle times within jobs and before job } \sigma_{\pi(f)}(i)}$$

The first part of the Eq. 27.7 calculates the idle time before the *first job* of the *f*-th family. For doing so, it uses information about the completion time of the previous family's *last job*, plus the setup time between the *f*-th family and its predecessor. The second part adds up the idle times between every two consecutive jobs of the *f*-th family, until job $\sigma_{\pi(f)}(i)$. The completion times of the previous job in the present machine (*m*) and of the present job in the previous machine (*m*-1) are utilized. The makespan is then calculated iteratively, job by job, machine by machine, being represented by the last job's completion time in the last machine.

In view of the NP-hard nature of the general FS problem, most researchers have focused on developing heuristic procedures that provide good permutation schedules (in which the order of job processing is the same on all machines) within a reasonable amount of computational time. However, there is no guarantee that a permutation schedule will be optimal when the shop contains several machines. In fact, it is very likely that the optimal schedule will have different job-permutations

[6] Idle times are periods when the machine is not operating because it is waiting for the next job to become available.

in each machine. However, considering different job-permutations increases the resulting computational complexity so much that the problem becomes intractable even for very small instances. The usual approach is then to assume the same job-permutation for all machines, reducing the problem's complexity.

Recent reviews (Allahverdi et al. 1999; Cheng et al. 2000) showed that most prior research on manufacturing cell scheduling has assumed sequence-independent setup times. For the flowshop manufacturing cell scheduling problem involving sequence-dependent setup times, (Hitomi et al. 1977) described a simulation model and showed that the scheduling rules considering explicitly sequence-dependent setups outperformed rules which did not explicitly do so. Realizing this, (Schaller et al. 2000) developed and tested several heuristic algorithms for minimizing the makespan in a FS with sequence-dependent family setup times.

27.4.1 The test instances

The instances utilized in this paper are the same presented in (Schaller et al. 2000) and are divided into three classes. In each class, processing times are random integers following a discrete uniform distribution DU(1, 10). As in the previously presented scheduling problems, the hardness depends on the balance between average processing times and the average setup times. Due to this processing times/setup times relation, three different classes of problems were used in the computational experiments. The setup times follow discrete uniform distributions in the ranges:

- Small setup times (SSU): DU(1, 20)
- Medium setup times (MSU): DU(1, 50)
- Large setup times (LSU): DU(1, 100)

According to these definitions, in the SSU-class instances, the ratio defined by average family setup time/average job processing time is approximately 2:1; in the MSU class, the ratio is 5:1, and in the LSU the ratio jumps to 10:1.

Moreover, problems were generated with the number of families varying between 3 and 10, and the number of jobs per family between 1 and 10. The number of machines varied between 3 and 10. For each combination of problem parameters, there are 30 problem instances. As an example of the notation, consider the *LSU108* set of problems: it consists of 30 instances with setup times in the [1, 100] interval, 10 families of jobs, 10 jobs per family at maximum and 8 machines. Considering all configurations tested, we obtain a total of 900 problem instances.

27.4.2 The memetic algorithm approach

The MA utilized for the FS problem follows the same structure of the PMS one. In order to describe the peculiarities of the algorithm for this specific problem, we will begin with the individual representation.

The FS scheduling problem has a structure that allows its division into two parts: the schedule of the families and the schedule of the jobs within the families. The representation adopted takes this division into account and is illustrated in Fig. 27.6. It consists of an arbitrary solution for a problem with twelve jobs and four families. Family 1 consists of jobs [1, 2, 3]; family 2 consists of jobs [4, 5, 6, 7]; family 3 of jobs [8, 9] and family 4 consists of jobs [10, 11, 12]. The processing sequence of the families is [1, 4, 2, 3]. Moreover, the jobs in family 1 are processed in the sequence [2, 1, 3]; family 2 in the order [7, 4, 6, 5], family 3 in the sequence [8, 9] and finally family 4 in the order [11, 12, 10].

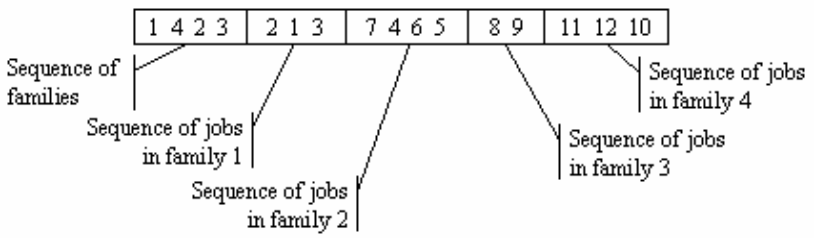

Fig. 27.6. Diagram of the individual representation utilized in the FS problem

The representation divides the solution into f+1 independent parts making the local search, crossover, mutation and other chromosome-level operators be executed separately within each part, without affecting the rest of the solution – which is reminiscent to divide-and-conquer strategies. The computational effort required by the local search is especially reduced due to this chromosome division.

The crossover utilized was the OX, already described in previous sections. The difference is that the OX is applied within each part of the chromosome, separately (see Fig. 27.7).

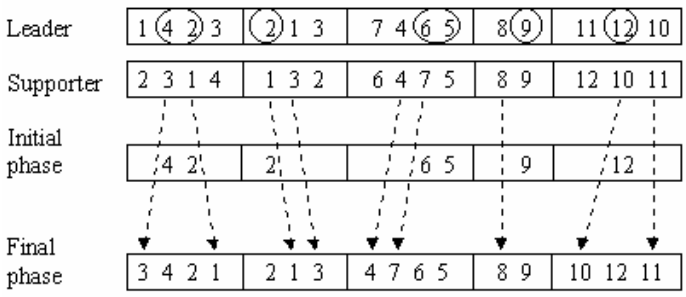

Fig. 27.7. Diagram of the OX crossover utilized in the FS problem

In Fig. 27.7, the OX crossover starts by selecting parts of the *leader*'s genetic material to be copied into the offspring. In the example, these parts are circled. This information is copied into the chromosome labeled as 'Initial phase'. Note that only one piece is copied from each part of the chromosome. In the *final*

phase, the *supporter* parent completes the offspring with its genetic information, with each part being filled from left to right, following the sequence of non-repeated alleles.

The local search scheme adopted is the same utilized in the PMS problem: *all-pairs* plus *insertion* neighborhoods, without neighborhood reduction. In this problem, the separation of the chromosome into parts has reduced the local search computational effort and no neighborhood reductions were necessary. Just to illustrate the effort-reduction effect when applying local search in an individual with five families and 10 jobs per family, the algorithm had in fact to apply the local search in six 10-allele sized individuals. A much easier task than applying a single local search in an individual with 60 alleles.

27.4.3 The flowshop computational results

The previous best results for the FS problem available in the literature were present in (Schaller et al. 2000). In that work, high quality lower bounds are provided, and the best heuristic for the problem is named CMD. It employs a dispatch rule together with a local search-based method.

The MA was programmed in Java JDK 2.0 and run using a Pentium II 266 MHz. Since the CMD is a constructive heuristic with a local search phase, it runs very quickly. The CPU times are below three seconds. For the MA, after a few considerations, we concluded that a 30-second limit was sufficient, given the instances' sizes. Therefore, the algorithm stops only if the lower bound is reached - i.e., the optimal solution is found – or after 30 seconds of CPU time. The Tables 27.5, 27.6 and 27.7 show the results for the MA and the CMD algorithm considering the LSU, MSU and SSU-type instances, respectively.

Table 27.5. Results for the LSU-type flowshop instances

	Memetic algorithm				CMD heuristic			
Instance	Min.	Ave.	Max.	CPU	Min.	Ave.	Max.	CPU
LSU33	0.00_{27}	0.07	1.12	3.1	0.00_{21}	0.91	8.41	0.04
LSU34	0.00_{20}	0.32	2.43	10.1	0.00_{12}	1.08	16.39	0.06
LSU44	0.00_{20}	0.20	1.09	10.1	0.00_{8}	1.95	10.27	0.12
LSU55	0.00_{18}	0.28	1.86	12.1	0.00_{4}	2.49	9.57	0.21
LSU56	0.00_{9}	0.51	2.42	21.1	0.00_{4}	3.37	17.07	0.26
LSU65	0.00_{15}	0.31	2.42	15.1	0.00_{3}	3.29	10.02	0.30
LSU66	0.00_{15}	0.19	1.36	15.3	0.00_{3}	3.03	10.47	0.44
LSU88	0.00_{6}	0.58	1.86	24.7	0.28_{0}	6.25	18.17	0.95
LSU108	0.00_{1}	0.47	1.19	29.8	0.12_{0}	6.22	11.25	1.82
LSU1010	0.00_{1}	0.77	2.27	29.5	0.53_{0}	6.30	11.42	2.37
Average	0.00_{132}	0.37	1.80	17.1	0.09_{55}	3.49	12.30	0.66

The results in Table 27.5 reveal an impressive performance for the MA. The figures represent percentage deviations from the lower bounds. In the 'Min'-labeled columns, the subscript figure is the number of times that the algorithm reached the lower bound, finding the optimal solution. In the 'Average' row, the subscript marks the sum of all optimal solutions found. The MA surpassed the CMD performance in all instance configurations. The large number of optimal solutions found is an indicative of the high quality of the lower bounds presented in (Schaller et al. 2000). We must emphasize that each instance configuration set was composed of 30 different instances. The total number of instances tested in each table is 300. Analyzing the MA against the CMD, the MA found 132 (44% of the entire set) optimal instances, more than twice the number found by the CMD. All the averages were better, except the CPU time, as expected. These results do not cause surprise, since the MA employs much more problem-driven features than the CMD algorithm.

An interesting characteristic is present in the results and is worth to be emphasized. The average percentage deviation from the lower bounds remains at low levels for all instances' sizes. Usually, the search methods lose performance for the larger instances, in an indication that they are gradually getting beyond the algorithm's search capabilities. This has occurred with the CMD algorithm, which began with an average deviation of 0.91% and ended with a 6.30% deviation. However, this was not observed for the MA, leading to the conclusion that probably the algorithm is still very far from its limit and would be able to deal with instances larger than the ones utilized. Although there is a slight increase in the MA's averages as the instances become larger, this is probably due to the declining lower bounds' quality.

Table 27.6. Results for the MSU-type flowshop instances

	Memetic algorithm				CMD heuristic			
Instance	Min.	Ave.	Max.	CPU	Min.	Ave.	Max.	CPU
MSU33	0.00_{23}	0.37	3.37	7.1	0.00_{21}	0.92	11.46	0.04
MSU34	0.00_{17}	0.56	2.29	13.1	0.00_{11}	2.00	16.28	0.05
MSU44	0.00_{11}	0.50	2.32	19.1	0.00_{4}	1.96	11.11	0.13
MSU55	0.00_{15}	0.45	2.09	15.1	0.00_{4}	3.10	8.48	0.19
MSU56	0.00_{6}	0.87	3.12	24.1	0.00_{1}	3.58	13.13	0.27
MSU65	0.00_{14}	0.36	1.22	16.2	0.00_{3}	3.68	8.88	0.30
MSU66	0.00_{8}	0.50	1.63	22.7	0.00_{1}	4.59	15.77	0.40
MSU88	0.00_{3}	0.99	2.98	27.3	0.63_{0}	5.68	12.68	0.97
MSU108	0.00_{1}	0.86	1.80	30.1	2.89_{0}	6.11	10.83	1.84
MSU1010	0.15_{0}	1.15	2.53	30.8	2.29_{0}	5.73	9.92	2.37
Average	0.01_{98}	0.66	2.33	20.6	0.58_{45}	3.73	11.85	0.65

The Table 27.6 shows a similar performance, with the MA surpassing the CMD in all criteria but the CPU time. The MA has reached fewer optimal solutions (98

in total, almost 33% of the total), but that is a number still two times larger than the one obtained by the CMD algorithm. The averages are a little worse than the ones for the LSU-type instances, but we believe this is due to the decreasing quality of the lower bounds. Both instance sets (LSU and MSU) have similar characteristics, such as number of machines and families. Moreover, the MA does not employ any procedure, like local search reduction policies, which could be affected by a change in the setup time's interval. Therefore, no reduction in the number of optimal solutions found should be observed at all. The lower bounds might in fact work better with larger setup times. This vision is reinforced by a reduction in the number of optimal solutions found by the CMD algorithm, too.

Table 27.7. Results for the SSU-type flowshop instances

	Memetic algorithm				CMD heuristic			
Instance	Min.	Ave.	Max.	CPU	Min.	Ave.	Max.	CPU
SSU33	0.00_{23}	0.31	2.47	7.1	0.00_{18}	0.67	4.13	0.04
SSU34	0.00_{15}	0.83	2.94	15.1	0.00_{8}	1.85	8.46	0.05
SSU44	0.00_{15}	0.57	2.90	15.0	0.00_{8}	1.94	8.33	0.12
SSU55	0.00_{4}	0.92	2.34	26.0	0.00_{1}	3.15	6.61	0.20
SSU56	0.00_{3}	1.56	3.08	27.4	0.00_{1}	4.02	8.93	0.25
SSU65	0.00_{6}	0.99	3.44	24.0	0.00_{2}	3.00	6.90	0.32
SSU66	0.00_{1}	1.28	2.72	29.1	1.24_{0}	4.06	9.16	0.44
SSU88	0.28_{0}	1.85	3.31	30.3	3.20_{0}	5.62	8.59	0.91
SSU108	0.72_{0}	1.77	2.90	30.7	2.58_{0}	5.63	8.96	1.82
SSU1010	0.59_{0}	2.33	3.65	30.6	4.07_{0}	6.86	9.11	2.21
Average	0.16_{67}	1.24	2.98	23.5	1.11_{38}	3.68	7.92	0.64

In Table 27.7, one can see that the MA maintained its behaviour, obtaining good general average performance followed by a decrease in the number of optimal solutions found. The number dropped to 22% of all instances. The loss of performance with the increasing instances' size is now clear, with the MA being unable to reach the lower bounds for instances with eight or more families of jobs.

Given previous experiences with local searches and how they behave with different sizes of solution spaces, it is likely that the number of optimal solutions found is much larger than the ones reported. Finding the optimal permutation in 10-allele sequences is an easy task when all-pairs and insertion neighbourhoods are employed.

27.5 Discussion

This chapter presented three job scheduling problems: Single Machine Scheduling (SMS), Parallel Machine Scheduling (PMS) and Flowshop Scheduling (FS). In order to deal with these problems a MA was employed. The algorithm was very

similar to that introduced in Chap. 18 for the Gate Matrix Layout problem. The MA had an impressive performance for the problems, what reinforces the belief that the algorithm's main features are general enough to deal with a much broader variety of problems.

In the SMS section, a procedure to transform Asymmetrical Travelling Salesman Problem's (ATSP) instances into SMS ones was described. By using it, one can create a SMS instance with known high-quality solutions if an ATSP instance with a known optimal tour is utilized as the starting point. Although this procedure is not yet proved to create optimal SMS instances, during the tests using TSPLIB instances, no counter-example was ever found. If the procedure is proved to be correct – or under which circumstances it is correct – it would solve a problem quite common is SMS tests: The lack of large optimal instances to test algorithms. In the literature, there are ATSP instances with thousands of cities solved to optimality. Such instances could be used to create SMS ones, also with thousands of jobs and known optimal solution. The procedure has some drawbacks, like a limitation on how the processing times and the due dates must be generated and the fact that the resulting instances are EDD/SPT-optimally solvable. Nevertheless, such instances can be very useful to test the performance of general-use metaheuristics.

The MA was able to find optimal solutions for instances with up to 100 jobs, with a high rate of success and CPU times no longer than four minutes, in average. The instances with 71 jobs or less had an impressive 100% rate of success. The 100-job instances also had a high rate of success, but apparently, they mark the beginning of the MA's search power exhaustion. For larger instances, new features will probably have to be added to the MA in order to sustain the performance, or more CPU time will have to be given.

In the PMS problem, the MA is compared to a well-tailored *tabu search* (TS), named *fast tabu*. The TS results were the best previously available in the literature (França et al. 1996) for the problem. Instances with up to 80 jobs and 8 machines were tested, as well as four setup times configurations. The MA performance was comparable to the TS especially for the so-called *structured* instances, which setup times follow the triangular inequality, and for problems with fewer machines, independently of the number of jobs. In this problem, the lack of a neighborhood-reduction policy has weighted against the MA. Although the performance was not disappointing, it could have been better with a reduction policy being applied. This is the logical next step for this problem. Both MA and TS were evaluated against lower bounds and optimal solutions for the smaller instances. For the larger instances, upper bounds obtained through a high-performance TS, named *long tabu*, were the benchmark performance measure. However, it is important to emphasize that the long tabu algorithm requires long CPU times, usually hours, against the few seconds or minutes required by the MA and the fast tabu.

The last problem addressed in the chapter was the flowshop scheduling (FS). In addition to the ordinary FS problem, this one also considers that the jobs are grouped in families, with jobs in the same family requiring similar tooling and machinery. That makes the setup times between jobs within the same family be very small, so that they can be added to the job's processing time. However, major

setup times are required to change the production from one family to another. The most remarkable feature of the MA for this problem was the representation utilized. It reduced the search space for the problem so much that even large-size instances, with 10 families of jobs and 10 machines were still below the MA's search capability.

Although its importance for the industry, this FS-variant has received little attention in the past, and the previous best method available in the literature was a constructive rule followed by a local search procedure, named CMD (Schaller et al. 2000). In this same paper, several lower bounds were calculated and utilized as benchmarks. The MA presented in this chapter is probably the first metaheuristic approach for it. The performance of the MA has dramatically surpassed the CMD algorithm, as was expected. But most impressive was the number of optimal solutions found, a situation resultant from the high quality of the lower bounds and the superior performance of the MA.

The MA utilized in all problems presented in this chapter is available in the NP-Opt Framework (Mendes et al. 2001). As said in the Chap. 18, the NP-Opt is an object-oriented, Java-based software, which is updated and improved continuously by a team of collaborators. For more information, please refer to the NP-Opt homepage[7], where the latest version is always available for download, as well as the software guide and the instances utilized in this chapter.

References

Allahverdi A, Gupta JND, Aldowaisan T (1999) A survey of scheduling research involving setup considerations. OMEGA – International Journal of Management Science 27:219–239

Baker KR (1974) Introduction to Sequencing and Scheduling. John Wiley & Sons, New York

Cheng TCE, Gupta JND, Wang G (2000) A review of flowshop scheduling research with setup times. Production and Operations Management 9:283–302

Dearing PM, Henderson RA (1984) Assigning looms in a textile weaving operation with changeover limitations. Production and Inventory Management 25:23–31

Du J, Leung JYT (1990) Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research 15:483–495

[7] http://www.densis.fee.unicamp.br/~smendes/NP-Opt.html.

França PM, Gendreau M, Laport G, Muller F (1996) A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. International Journal of Production Economics 43:79–89

França PM, Mendes AS, Moscato P (2001) A Memetic Algorithm for the total tardiness Single Machine Scheduling Problem. European Journal of Operational Research 132:224–242

Frederickson G, Hecht MS, Kim CE (1978) Approximation algorithm for some routing problems. SIAM Journal on Computing 7:178–193

Gen M, Cheng R (1997) Genetic Algorithms and Engineering Design. John Wiley & Sons, New York

Goldberg DE (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley

Graham RL, Lawler EL, Lenstra JK, Rinooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5:287–326

Graves SC (1981) A review of production scheduling. Operations Research 29:646–675

Gupta JND, Darrow WP (1986) The two-machine sequence dependent flowshop scheduling problem. European Journal of Operational Research 24:439–446

Hitomi K, Nakamura N, Yoshida T, Okuda K (1977) An Experimental Investigation of Group Production Scheduling. Proceedings of the 4th International Conference on Production Research, pp 608–617

Lawler EL, Lenstra JK, Rinooy Kan AHG, Shmoys DB (1993) Sequencing and Scheduling: Algorithms and Complexity. In: Handbooks in Operations Research and Management Science Vol. 4. North-Holland, pp 445–522

Lee YH, Bhaskaran K, Pinedo M (1997) A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transactions 29:45–52

Mendes AS, França PM, Moscato P (2001) NP-Opt: An Optimization Framework for NP Problems. Proceedings of the POM2001 – International Conference of the Production and Operations Management Society, pp 82–89

Mendes AS, Muller F, França PM, Moscato P (2002) Comparing Meta-Heuristic Approaches for Parallel Machine Scheduling Problems. Production Planning and Control 13:143–154

Ow PS, Morton TE (1989) The single machine early/tardy problem. Management Science 35:177–191

Ragatz GL (1993) A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence dependent setup times. Proceedings of the 24th Annual Meeting of the Decision Sciences Institute, pp 1375–1377

Raman N, Rachamadugu RV, Talbot FB (1989) Real time scheduling of an automated manufacturing center. European Journal of Operational Research 40:222–242

Rubin PA, Ragatz GL (1995) Scheduling in a sequence dependent setup environment with genetic search. Computers & Operations Research 22:85–99

Schaller JE, Gupta JND, Vakharia AJ (2000) Scheduling a Flowline Manufacturing Cell with Sequence Dependent Family Setup Times. European Journal of Operational Research 125:324–339

Sumichrast R, Baker JR (1987) Scheduling parallel processors: an integer linear programming based heuristic for minimizing setup time. International Journal of Production Research 25:761–771

Tan KC, Narasimhan R (1997) Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach. OMEGA – International Journal of Management Science 25:619–634