Annals of Operations Research 63(1996)397–414

A combined branch-and-bound and genetic algorithm based approach for a flowshop scheduling problem

Amit Nagar
School of Business, University of Windsor, Windsor, Ontario, Canada

Sunderesh S. Heragu and Jorge Haddock
Decision Sciences and Engineering Systems Department, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

In this paper, we study the application of a meta-heuristic to a two-machine flowshop scheduling problem. The meta-heuristic uses a branch-and-bound procedure to generate some information, which in turn is used to guide a genetic algorithm's search for optimal and near-optimal solutions. The criteria considered are makespan and average job flowtime. The problem has applications in flowshop environments where management is interested in reducing turn-around and job idle times simultaneously. We develop the combined branch-and-bound and genetic algorithm based procedure and two modified versions of it. Their performance is compared with that of three algorithms: pure branch-and-bound, pure genetic algorithm, and a heuristic. The results indicate that the combined approach and its modified versions are better than either of the pure strategies as well as the heuristic algorithm.

Keywords: Genetic algorithm, scheduling, branch-and-bound.

1. Introduction and brief literature review

We study a two-machine bicriteria flowshop scheduling problem in which the objective is to minimize the weighted sum of schedule makespan and average job flowtime. The problem has applications in industries where each job must undergo two basic processes in the same sequence. For example, in the steel industry, each job undergoes wire-drawing first and annealing next. At the aggregate planning or macro level in general manufacturing industries, each job must undergo fabrication first and assembly next. The two-machine bicriteria flowshop scheduling problem has been studied in [1–3]. All the previous approaches to this problem have relied on conventional optimization techniques such as goal programming and branch-and-bound. Selen and Hott [1] developed a goal programming formulation for the bicriteria

problem. An improved model was later developed by Wilson [2], who used substantially fewer variables at the expense of additional constraints. A drawback of both the models is the presence of n^2 integer variables, where n is the number of jobs. Obviously, this restricts the applicability of the models to problems of small size. Recently, Rajendran [3] developed a branch-and-bound approach to solve this bicriteria problem. His approach was to minimize the total flowtime subject to obtaining optimal makespan. Since the branch-and-bound approach was practical only for small problems, he developed two heuristic procedures based on the adjacent pairwise job interchange and heuristic preference relations. For a recent literature review of models and algorithms for multiple and bicriteria scheduling, refer to Nagar et al. [4].

In this paper, we present a new algorithm which combines techniques from artificial intelligence (AI) with traditional operations research (OR) procedures. In particular, a genetic algorithm (GA) is combined with a branch-and-bound algorithm to form a hybrid algorithm. The algorithm is referred to as the meta-heuristic algorithm throughout this paper. The main benefit of combining the techniques from the two domains is that the resulting meta-heuristic draws from the good characteristics of both. The branch-and-bound algorithm is used to generate information that tells us whether or not certain partial schedules when completed into full ones will yield optimal solutions. For example, in a problem with six jobs, if we know that the lower bound of a partial schedule [1,3,4] is greater than a known upper bound, we need not evaluate all the schedules which are completions of this partial schedule. Thus, we need not evaluate {1, 3, 4, 2, 5, 6}, {1, 3, 4, 2, 6, 5}, {1, 3, 4, 5, 2, 6}, {1, 3, 4, 5, 6, 2}, {1, 3, 4, 6, 2, 5} and {1, 3, 4, 6, 5, 2}. If the GA's search for a near-optimal solution is guided by providing such information and thus directed away from known sub-optimal areas, its accuracy and computational efficiency can be improved. This is the central idea presented in our paper and is discussed in more detail in section 3. We also propose two modifications to the meta-heuristic. Using test problems, we show that the meta-heuristic or its modifications perform well on all three counts – solution quality, computational time, and the problem size which can be solved.

GAs are stochastic search algorithms designed to search large and complex nonlinear spaces. They were first developed by Holland [5] in the 1970s. As the name suggests, GAs were motivated by the theory of adaptation and evolution in biological systems. Due to the robust nature of these algorithms, they have been successfully applied to many combinatorial problems including: quadratic assignment, traveling salesman, and scheduling problems [6]. Applications of the basic GA to solve specific scheduling problems can be found in [7–10], amongst others. To improve the performance of GAs on scheduling problems, a number of papers have suggested use of specific operators. For example, [11–13] have designed recombination operators specific to the problem considered.

To the best of our knowledge, GA has not been applied to the problem considered in our paper. Moreover, no other technique combines traditional OR with AI-based techniques as we do. Although Syswerda and Palmucci [14] have briefly mentioned

that providing domain knowledge to GA will make the latter more powerful, they have not provided any formal, efficient mechanisms for doing so.

The next section introduces the problem studied in this paper. In section 3, we discuss the meta-heuristic and its modifications. Computational results with the algorithms are presented in section 4. Summary and conclusions are provided in the last section.

2. Problem statement

The scheduling problem considered in this paper is a two-machine flowshop with schedule makespan and average job flowtime as the two criteria. In scheduling terminology, the problem can be presented as $n/m = 2/\text{Flowshop}/C_{max}, \overline{F}$, where n is the number of jobs, m is the number of machines, $\overline{F}$ is the average flowtime, and C_{max} represents makespan. Flowshop is a shop configuration in which machines are arranged serially with unidirectional work flow. The jobs are two-stage in nature and are to be processed in the same order on each machine. The following assumptions characterize a flowshop:

- All jobs are available for processing at time zero.
- Setup times are known and included in the processing times.
- Preemption is disallowed and machines are continuously available.
- There is an infinite buffer between the two machines.

Job flowtime is defined as the time spent by the job in the system. To obtain an estimate of the total duration of process or turn-around time, makespan is employed as a criterion. It is the time at which the last job completes its processing on the last machine. The objective function is a linear combination of the two criteria and is denoted by $(\alpha \overline{F} + \beta C_{max})$, where $\alpha + \beta = 1$, $\alpha \geq 0$, $\beta \geq 0$. It should be noted that for a two-machine flowshop, the bicriteria problem is easily solved when $\alpha = 0$ (using the well-known Johnson algorithm) but is NP-hard when $\alpha = 1$. Hence, a problem with an α value closer to zero is likely to be easier than one for which it is closer to one [15].

One of the primary objectives is a Just-in-Time (JIT) system to minimize all forms of waste. Job idle time and work in progress (WIP) inventory are, of course, two such forms. By minimizing average job flowtime and schedule makespan, we indirectly minimize the total job idle time and WIP. Thus, the above objective is useful in a JIT environment.

There are primarily two methods of dealing with multiple criteria in a model. The first is called a posteriori articulation of preferences, and it necessitates the determination of an efficient frontier which provides the decision maker with efficient solutions (for example, see [16]). S/he is then required to choose a solution based on the trade-offs. However, as the problem size increases, the information available to the decision maker explodes, thereby limiting his/her ability to identify the most

satisfactory solution. The second method, called the a priori articulation of facts, requires the decision maker to assign weights for the two criteria. The choice of each technique is guided by the amount of information available to the decision maker. As the problem size increases, the second method is more useful and practical than the first and hence is adopted in this paper.

3. Description of the meta-heuristic and its modifications

Different regions of the solution space can be defined by fixing certain schedule positions. For example, in a 3-job problem, {1, *, *}, {2, *, *}, {*, *, 2} are three of the many possible regions. The symbol * is generally referred to as a "don't care" symbol in GA terminology. Using information concerning the solution space's regions and guided by two operators – reproduction and crossover – a GA searches for optimal or near-optimal solutions.

The reproduction operator samples the individual schedules in proportion to their objective function values (OFVs). A schedule with an OFV higher than the population average has a greater probability of being sampled than schedules with lower OFVs (for a problem with maximization objective). By explicitly testing the goodness of a *single* schedule, it is clear that GA implicitly tests a number of different *regions* determined by the job positions in the schedule. In other words, if a schedule {1, 3, 2} is sampled because its OFV is higher than the population average, it implies that the GA has implicitly tested the regions {1, *, *}, {*, *, 2}, amongst others.

Reproduction alone cannot guide GA to the optimal solution. In the absence of other operators, it will simply give us the best schedule from the initial populations without exploring other regions of the solution space. To provide diversity to GA's search, the crossover operator is used. It is a matching and swapping operator which combines the partial solutions from two "parent" schedules to produce two new "offsprings" schedules. Crossover performs two functions:

- samples new schedules from the target area (Exploitation), and
- directs the search into new regions of the solution spaces (Exploration).

To elaborate on the exploitation phase, consider a schedule {1, 3, 2} sampled by the reproduction operator in the first generation. Now this schedule is a member of six different target regions, namely, {1, *, *}, {1, 3, *}, {*, 3, *}, {1, *, 2}, {*, *, 2}, {*, 3, 2}. Note that each of these regions defines a subset of schedules, for example, {1, *, *} defines the subset {1, 2, 3}, {1, 3, 2}. GA has identified the above as good regions based on only one sample point. It is the role of the crossover operator to provide more samples of these regions from the subset of schedules defined by each. This will enable GA to evaluate the target regions based on a large number of samples. Thus, the final evaluation of the goodness of the regions has a sound basis

and is not a chance occurrence. In its exploratory phase, GA destroys certain partial solutions, for example {1, *, 2} and forms new partial ones. This allows it to guide its search into new regions of the solution space. Obtaining a balance between exploration and exploitation is a fundamental problem in the theory of adaptation.

Thus, it is a combination of exploration and exploitation that allows GA to sample already targeted regions to confirm or disprove the previous estimates while exploring new regions in the solution space. GA attempts to ensure that there is an optimal allocation of jobs to schedule positions. This in turn improves the possibility of obtaining a near optimal schedule at the end of a run.

3.1. RELATIONSHIP BETWEEN BRANCH-AND-BOUND AND GENETIC ALGORITHM SEARCH

The branch-and-bound algorithm used in the meta-heuristic is discussed in detail in [15]. It uses a result that states the global lower bound can be obtained from the sequence in which jobs are arranged in shortest processing time (SPT) order on the second machine. To tighten the lower bound, appropriate idle times are included. The branch-and-bound algorithm uses the depth-first search strategy, since it requires very little storage space and hence can be used for larger problems. It initializes the tree with an initial feasible solution and an upper bound, both of which are obtained using a heuristic algorithm. The heuristic is a construction algorithm that adds jobs to the schedule, one at a time, so that the job idle time is minimized. At each stage of schedule building, it chooses the unscheduled job having the earliest completion time on the second machine as the next job for processing. This procedure is continued until all jobs have been scheduled. The underlying principle of the heuristic is to "pull back" the objective function at each stage of the schedule generation. To further improve the solution, it performs two-opt and three-opt job interchanges. The heuristic provides reasonably tight upper bounds and allows a large number of suboptimal solutions to be fathomed. In fact, it compares well with the branch-and-bound algorithm itself for some test problems and can be used to solve large-scale problems.

To understand functioning of the branch-and-bound algorithm, consider figure 1. A schedule is generated at each node of the ith level, $i = 1, 2, \ldots, n$, by assigning jobs to the first i schedule positions and arranging the remaining jobs according to their SPT sequence. Each node in the branch-and-bound tree corresponds to a partial schedule. As we traverse downwards along the tree, the number of jobs whose positions are known increases. The branch-and-bound algorithm computes the lower bound for each node, compares it with the upper bound and determines whether the nodes are to be fathomed or not. It performs well for the following two types of problems:

(i) problems in which the second machine is dominant, i.e. for each job, the processing time on the second machine is greater than on the first, and

(ii) problems in which the processing times are the same on both machines for each job.

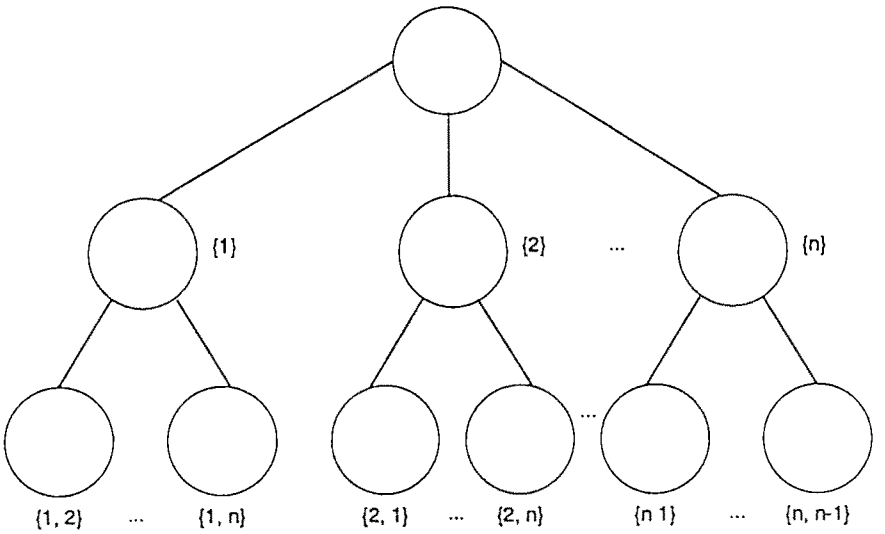

Figure 1. A branch-and-bound tree.

As demonstrated in [15], the branch-and-bound algorithm has solved type (i) problems having up to 500 jobs. For type (ii) problems, it is shown that the heuristic used for determining an upper bound by itself provides optimal solutions. However, for general problems in which the above two conditions do not hold, or when the first machine is dominant or the input data is such that there is much idle time, the performance of the branch-and-bound algorithm deteriorates significantly. It is for the general problem that we develop the meta-heuristic. In a broad sense, the meta-heuristic works as follows. The branch-and-bound algorithm is allowed to evaluate nodes until a certain depth is reached in the tree. Their lower bounds are then compared to a known upper bound, and node fathoming decisions are made. If a node can be fathomed, then obviously each completion of the partial schedule corresponding to the fathomed node can only produce a sub-optimal solution. Using such information, the GA is prohibited from searching in known sub-optimal regions, to increase its accuracy and efficiency. This is the central idea of the meta-heuristic. It should be noted that the efficiency of GA in the meta-heuristic depends upon the information made available by the branch-and-bound algorithim, which itself depends upon the node level to which it is allowed to proceed. Thus, an increase in the GA's efficiency comes at the expense of more computational effort by the branch-and-bound algorithm. Hence, we need to carefully determine the level to which the latter is allowed to proceed. We now provide steps in the meta-heuristic and subsequently discuss each in detail.

META-HEURISTIC ALGORITHM

Step 1: Using the heuristic algorithm in [15], obtain an upper bound and initialize the population pool.

Step 2: Fix a level to which the branch-and-bound algorithm is allowed to proceed and obtain node fathoming information.

Step 3: Generate an initial population from the pool in step 1.

Step 4: Use the roulette wheel selection procedure to choose schedules for the next generation.

Step 5: Select a pair of schedules from the population. Go to step 6.

Step 6: If the schedules are known to be suboptimal (based on node information obtained from step 2), go to step 7. Otherwise, go to step 8.

Step 7: With a pre-specified probability p_d, disrupt the schedule using the SSD operator.

Step 8: With a pre-specified probability p_c, perform crossover using the PMX operator.

Step 9: With a pre-specified probability p_m, perform mutation by swapping a randomly selected pair of jobs.

Step 10: If all population members have been evaluated, go to step 11. Otherwise, go to step 5.

Step 11: If the algorithm has run for the specified number of generations, STOP. Otherwise, go to step 4.

To obtain the initial population, most GAs randomly generate the required number of solutions (schedules). However, in our implementation, we generate a large part of the initial population using the heuristic that provides us with the upper bound in the branch-and-bound algorithm. To do so, we first obtain a schedule using the heuristic. We then explore the neighborhood of this schedule and retain those with an OFV within a certain percentage of the heuristic schedule's OFV. (The neighboring schedules are generated by performing pairwise and 3-opt job interchanges.) If the initial population consists entirely of schedules generated using a local minimum algorithm, it is highly likely that the GA search will converge prematurely, prior to finding the global optimum. To avoid such an occurrence and maintain enough diversity in the population, we generate a percentage of the population randomly.

Then, the branch-and-bound algorithm is invoked. Obviously, its computational effort increases significantly if it is allowed to evaluate a large number of levels in the tree. On the other hand, if it is limited to a few levels, then there is not much information available to guide the GA's search. Thus, we must limit its search up to a level based on the problem size.

In the traditional or generic GA, a pair of selected schedules is used by the crossover operators to generate offsprings. Intuitively, it appears that if we generate offsprings selectively from "good" parents, the quality of the resulting offsprings is likely to be superior. This is exactly what is done in step 6 of the meta-heuristic. We ensure that each schedule selected in step 5 is one that is not known to be in the sub-

optimal region (step 7). If it is, then we disrupt job positions in this schedule by means of a disruption operator. This disruption operator is termed as a static schedule disruption operator (SSD). The static nature of the operator is explained later in this section. The disruption is carried out by interchanging two jobs in the schedule. We select the first job in the schedule as one of the jobs for interchange. Notice from figure 1 that only the first schedule position is fixed for nodes at the first level of the tree. Thus, the first-level node i corresponds to job i. By replacing the job in this position, we are selecting a new first-level node of the tree. To select the second job, we compute the upper bounds for all the first-level nodes of the tree. The roulette wheel selection (RWS) procedure is then used to select a node using the upper bound information. It is designed to ensure that the node with the least upper bound has the largest probability of being selected. The node selected through the RWS procedure provides us with the required second job (due to the one-to-one correspondence between nodes and jobs). The second job is then interchanged with the first job to complete the disruption process. Intuitively, this process of selecting the second job is expected to direct the genetic algorithm search towards the potentially optimal branches.

Once the known sub-optimal schedules have been disrupted, the GA uses a crossover operator to produce offsprings from parents. An important requirement of the crossover operator for ordering problems is to maintain feasibility. While a number of crossover operators such as cycle, order, etc., are available, we use the partially matched crossover (PMX) operator. It is a matching and swapping operator which performs these operations on a pair of selected schedules. PMX is a 2-point crossover operator. It works by selecting two points within the schedule. The jobs within the points are swapped (invariably leading to two infeasible schedules). A mapping operation is then used to remove this infeasibility from each of the resulting schedules. Consider the following schedules (adapted from [6]):

Parent 1: 9 8 4 {5 6 7} 1 3 2 10
Parent 2: 8 7 1 {2 3 10} 9 5 4 6

The crossover points are indicated by the braces "{". Swapping jobs between the crossover points results in two infeasible schedules, namely,

Offspring 1: 9 8 4 {2 3 10} 1 3 2 10
Offspring 2: 8 7 1 {5 6 7} 9 5 4 6

For offspring 1, the mapping operation maps the jobs 2, 3, 10 outside the interval to jobs 5, 6 and 7, respectively. A similar operation is carried out for the second offspring. Thus, the following schedules are obtained after the crossover operation:

Offspring 1: 9 8 4 2 3 10 1 6 5 7
Offspring 2: 8 10 1 5 6 7 9 2 4 3

Upon completion of a generation, reproduction is used to generate a new population for the next generation. This is done by selecting schedules in proportion to their OFVs. A number of ways have been suggested in the literature for implementation of this operator. We use the RWS procedure for reproduction. Also, GAs are designed to maximize the objective function. However, our objective function is a weighted sum of average flowtime and makespan which is required to be minimized. Therefore, we transform the objective function of this problem to a maximization function given by $e^{(\alpha \bar{F} + \beta C_{max})/100}$. This transformation meets the genetic algorithm requirement of maximization while introducing more sensitivity into the objective function.

Conceptually, the meta-heuristic differs from the conventional GA in its interface with the branch-and-bound algorithm. The meta-heuristic uses a schedule disruption operator to efficiently guide its search by using the available information. The remaining operators for the meta-heuristic are similar to the ones in a conventional GA. We now discuss two modifications to the meta-heuristic.

MODIFICATION 1

In developing the schedule disruption operator for the meta-heuristic, we relied on static worst-case information about the objective function, i.e. information collected at the beginning of the run (hence the name static schedule disruption operator). This was then used by the embedded GA to disrupt known sub-optimal schedules. This seems to be a reasonable approach, since we are directing the GA towards certain "desirable" branches of the tree. However, it is also possible to develop a dynamic operator which is based on the form of the objective function. The objective function for this problem can be written as:

$$Z(S) = \alpha \bar{F}(S) + \beta C_{max}(S)$$

$$= \alpha \left[\frac{1}{n} \sum_{i=1}^{n} (n - i + 1)\{t_{i,2} + M_i\} \right] + \beta \left[\sum_{i=1}^{n} \{t_{i,2} + M_i\} \right],$$

where

S = schedule,

$\bar{F}(S)$ = average flowtime for schedule S,

$C_{max}(S)$ = makespan for schedule S,

α, β = weights for $\bar{F}(S)$ and $C_{max}(S)$,

M_i = machine idle time before processing job scheduled in ith position and after processing job scheduled in $(i-1)$th position on second machine,

$t_{i,2}$ = processing time of job scheduled in ith position on second machine.

A good schedule corresponding to this objective function is more likely to have jobs with smaller processing times at the beginning rather than at the end of

the schedule. Also, smaller values of idle time are more desirable at the beginning of a schedule. This is because the weights are arranged in decreasing order of magnitude. These observations are used to develop the new dynamic schedule disruption (DSD) operator. The DSD operator is dynamic in the sense that it uses current schedule information rather than information collected at the beginning of the run (as is done by the SSD operator). It randomly selects two jobs i, j in the current schedule and determines the sum of processing times (on the two machines) for each job. If i is positioned before j and $t_{i,1} + t_{i,2}$ is greater than or equal to $t_{j,1} + t_{j,2}$, the positions of the two jobs i, j are interchanged. In other words, amongst the two randomly selected jobs, it places the one with the smallest sum of processing times first. Thus, the DSD operator attempts to disrupt the current (sub-optimal) schedule and find an improved one, if possible. This operator is used in modification 1 instead of the SSD operator used in the meta-heuristic. Apart from this change in step 7, modification 1 is similar to the original meta-heuristic.

MODIFICATION 2

The meta-heuristic works well when the embedded branch-and-bound algorithm can provide sufficient information to direct the GA from known sub-optimal regions. However, in large problems (i.e. problems with 500 or more jobs) or those in which the processing time data are such that there is much idle time even in optimal/near-optimal schedules, the branch-and-bound algorithm cannot provide much node fathoming information since the gap between lower and upper bounds is large. As a result, it cannot provide sufficient problem-specific information to the embedded GA. Using the average dominance concept, Nagar et al. [15] demonstrate the conditions under which the branch-and-bound algorithm is able to fathom a large percentage of the nodes. Average dominance is defined as the sum of processing times of all jobs on the first machine minus that on the second machine. When the average dominance values are close to zero, the problem is difficult for branch-and-bound. Thus, for such problems, the meta-heuristic is likely to perform poorly. Hence, we have developed another modified version (modification 2) of the meta-heuristic which does not use any information from the branch-and-bound procedure. Instead, modification 2 uses the worst half of the population for selecting the schedules for disruption. In each generation, the schedules are arranged in descending order of their OFVs. The worst half then corresponds to the first half of the population. Modification 2 uses the DSD operator for disrupting schedules in the first half of the population. Such a procedure is likely to perform better in the absence of adequate information from the branch-and-bound algorithm.

In summary, modification 1 uses the branch-and-bound algorithm to obtain node fathoming information but employs the DSD operator to disrupt sub-optimal schedules. On the other hand, modification 2 does not use the branch-and-bound algorithm, but uses the DSD operator to disrupt schedules that lie in the worst half of the population.

4. Experimental results

In this section, we present the results of our experimentation with the meta-heuristic and its two modifications. Where possible, the results are compared with the optimal solutions provided by the branch-and-bound algorithm. For larger problems, the heuristic algorithm discussed in section 3 and the conventional or genetic GA are used for comparison purposes.

In order to focus on the procedures developed in this paper, the design of experiments is such that more emphasis is placed on parameters such as problem size, probability of schedule disruption, and relative importance of the two criteria as determined by α. GA-specific parameters were held fixed for all the runs. Their values were determined through preliminary experimentation with GA. These parameters and their values are shown below:

population size,	pop size	=	100, 200
crossover probability,	p_c	=	0.9
mutation probability,	p_m	=	0.001
number of generations,	gen	=	200, 300

Problem size: Problems with sizes 10, 14, 20, 30, 50 and 100 were used for experimentation. The smaller population size was used for 10-, 14-job problems, whereas the larger one was used for 20-, 30-, 50- and 100-job problems.

Probability of schedule disruption: The probability of schedule disruption reflects the rate at which the known sub-optimal schedules are disrupted by SSD/DSD operators. Apart from the extreme values of 0 (no disruption) and 1 (always disruption), we also used disruption rates of 0.3 and 0.7.

α values: This parameter is important from the scheduler's perspective. It measures the relative importance of average flowtime and makespan in the objective function. We ran all the problems with the following two α values: 0.5 and 1.0. In addition, the small and medium sized problems, namely, 10-, 14-, 20- and 30-job problems were run for α values of 0.0. However, because the $\alpha = 0$ case can be optimally solved using Johnson's algorithm, results for this value of α are not provided. In fact, for all disruption probabilities, optimal solutions were obtained for the $\alpha = 0$ case for the 10- and 14-job problems. The $\alpha = 0.2$ case is also a relatively easy problem and hence is used for the 50- and 100-job problems only. The $\alpha = 0.5$ and 1.0 cases are of interest since they include a substantial flowtime component and hence render the problem difficult. Therefore, we provide results for all problem sizes for these two cases.

Table 1

Objective function value and CPU times for 10-job problems using various algorithms.

α	Data set	p_d	Meta-heuristic	Modification 1	Modification 2	Heuristic	Optimal (lower bound)
	1	0	374, 18.4	376, 11.8	377, 11.8		
		0.3	375, 17.3	379, 10.8	377, 12.5		
		0.7	374, 18.4	383, 9.6	377, 12.7	389, 0.4	374, 23 (279)
		1	375, 18.4	377, 9.4	376, 12.6		
	2	0	280, 18.3	282, 11.4	278, 16.5		
		0.3	282, 18.4	280, 12.6	279, 15.8		
1		0.7	279, 17.5	279, 13.0	279, 16.6	290, 0.5	276, 0.7 (240)
		1	281, 17.6	279, 11.3	279, 16.7		
	3	0	290, 20.6	–	290, 18.7		
		0.3	290, 20.1	–	290, 18.5		
		0.7	291, 19.7	–	290, 26.4	293, 0.5	290, 31.3 (220)
		1	291, 19.8	–	290, 18.9		
	1	0	224.5, 18.5	225.5, 13.9	225, 16.5		
		0.3	225, 16.4	224.5, 14.2	224.5, 16.5		
		0.7	225, 17.6	224, 14.5	225, 15.9	233, 0.6	224, 31.6 (172)
		1	226, 18.2	225, 13.6	225, 16.3		
	2	0	169, 17.5	172, 10.2	170.5, 16.2		
		0.3	169.5, 17.8	171, 11.8	169, 16.7		
0.5		0.7	169.5, 17.8	170, 13.6	169, 16.2	177, 0.5	168, 0.8 (149)
		1	170.5, 17.9	170.5, 11.6	170.5, 16.4		
	3	0	173.5, 20.4	–	173.5, 20.4		
		0.3	174.5, 20.6	–	173, 18.7		
		0.7	174, 20.2	–	173, 19.0	175, 0.4	173, 39.3 (135)
		1	174, 20.0	–	173.5, 19.2		

Tables 1 and 2 provide detailed results for the 10- and 14-job problems. They provide OFVs for the solutions generated by all the algorithms (meta-heuristic, its two modifications, the heuristic, and branch-and-bound) and the corresponding CPU times.

There is a wide variation in the CPU time for the branch-and-bound algorithm. For the problems tested, it requires anywhere between 0 and 5000 seconds. Since most of the problems in the two tables have been optimally solved by the branch-and-bound algorithm, it allows us to compare OFVs of the solutions generated by the meta-heuristic and its modifications with the optimal OFV. Their performance is also compared with a lower bound and the heuristic in tables 1 and 2. Our primary reason for including modification 1 in the comparison is to determine whether the

Table 2

Objective function value and CPU times for 14-job problems using various algorithms.

α	Data set	p_d	Meta-heuristic	Modification 1	Modification 2	Heuristic	Optimal (lower bound)
	1	0	485, 21.4	500, 19.4	487, 19.8		
		0.3	498, 21.2	493, 18.7	489, 20.3		
		0.7	490, 21.4	489, 21.3	487, 19.0	514, 0.3	475, 16.9 (412)
		1	496, 20.4	490, 20.3	486, 21.1		
	2	0	516, 25.9	–	515, 20.5		
		0.3	515, 23.2	–	516, 21.3		
1		0.7	516, 22.3	–	516, 21.5	516, 0.5	515, 6.5 (483)
		1	516, 24.6	–	516, 21.8		
	3	0	568, 21.0	564, 22.7	566, 19.9		
		0.3	579, 22.5	556, 20.5	565, 19.9		
		0.7	573, 22.4	563, 21.0	563, 21.5	592, 0.4	546, 33.0 (432)
		1	567, 29.3	563, 22.1	569, 21.1		
	1	0	288, 20.8	298, 21.0	288, 20.8		
		0.3	287.5, 22.3	298.5, 22.3	284, 19.9		
		0.7	294.5, 21.1	299.5, 21.9	284, 20.5	302, 0.6	278, 14.5 (246)
		1	292.5, 20.4	295.5, 21.6	289, 20.3		
	2	0	300.5, 19.2	–	300.5, 21.1		
		0.3	300.5, 19.9	–	300.5, 21.2		
0.5		0.7	300.5, 20.1	–	300.5, 21.1	301, 0.5	300, 0.3 (293)
		1	300.5, 19.5	–	300.5, 21.7		
	3	0	317, 22.1	322, 20.5	320.5, 19.4		
		0.3	326.5, 20.7	323, 21.8	319, 19.9		
		0.7	327.5, 20.6	323, 22.8	323, 19.9	340, 0.5	–, 5000[a] (253)
		1	326.5, 22	320.5, 19.9	317, 20.3		

[a] Even after approximately 5000 seconds, the problem could not be solved.

DSD operator (used in modification 1) has any advantage over the SSD operator (used in the meta-heuristic). Note that modification 1 is applied selectively to problems in which the average dominance is not close to zero, i.e. those problems for which the branch-and-bound algorithm provides abundant information for the embedded GA. This enables us to accurately compare the effect of the static and dynamic operators. However, based on the results for the meta-heuristic and modification 1, no conclusions can be made about the effectiveness of the two operators. Although intuitively the DSD operator appears to be superior to SSD, the computational results do not demonstrate this. The meta-heuristic and modification 1 out-perform each other over different data sets. As a result, modification 1 is excluded for larger problems in table 3.

Table 3

Objective function value and CPU times for medium and large problems using various algorithms.

Problem size	α	Data set	Meta-heuristic	Modification 2	Generic	Heuristic
20	1.0	1	651.75, 51.1	652.6, 28.7	–	695, 0.5
		2	481, 65.4	652.6, 28.7	–	489, 0.6
		3	562.8, 56.7	557.9, 41.4	–	584.5, 0.5
		4	606.75, 48.3	603.8, 40.2	–	626, 0.5
		5	2325.4, 66.5	2313.5, 49.2	–	2462.5, 0.6
	0.5	1	1178.25, 53.4	1172, 45.3	–	1256, 0.4
		2	861, 63.6	857.25, 64.2	–	875, 0.7
		3	1030.8, 56.2	1001, 49.1	–	1052, 0.6
		4	1091.8, 56.2	1081.25, 47.9	–	1127, 0.6
		5	4163.25, 66.8	4122.75, 64.2	–	4390, 0.5
30	1.0	1	1943.25, 27.05	1926.25, 20.4	2135, 60.9	1965, 0.7
		2	2255, 26.95	2241.5, 23.8	2459, 60.2	2303, 0.8
		3	2122, 26.68	2080, 25.1	2281, 62.1	2165, 0.8
		4	2255.25, 26.18	2319, 24.15	2522, 60.7	2419, 0.7
		5	2115, 26.33	2111.75, 24.1	2256, 60.2	2210, 0.7
	0.5	1	1049.38, 26.7	1043.88, 24.8	1186.5, 65.2	1061.5, 0.7
		2	1214.25, 27.3	1210.25, 24.0	1320.5, 67.4	1240.5, 0.8
		3	1134.75, 26.6	1131, 24.4	1213, 68.1	1166, 0.8
		4	1248.25, 25.3	1244.13, 24.6	1338.5, 65.8	1299.5, 0.8
		5	1140.8, 25.9	1134.13, 24.4	1221, 67	1189.5, 0.7
50	1.0	1	5631, 31.4	5583.5, 20.4	6399, 61.8	5793, 1.2
		2	6456.5, 34.0	6406, 28.0	7128, 58.4	6672, 1.4
		3	5543, 32.4	5551.5, 28.2	5933, 65.0	5817, 1.6
		4	6353.5, 34.5	6328.5, 33.4	6573, 65.8	6514, 1.1
		5	5402, 29.7	5397.5, 29.3	5814, 58.0	5486, 1.3
	0.5	1	2945.13, 33.6	2937.62, 30.0	3299.5, 53.8	3034, 1.3
		2	3365.63, 30.5	3353.88, 30.0	3773, 53.8	3485.5, 1.4
		3	2900.13, 31.4	2905.13, 28.7	3115, 52.6	3043.5, 1.4
		4	3297, 35.3	3333.5, 34.2	3499, 54.2	3406, 1.5
		5	2851, 32.5	2838, 31.3	3067.5, 52.7	2876.5, 1.6
	0.2	1	1353.6, 32.3	1344.9, 31.8	1489.4, 51.9	1377.8, 1.3
		2	1524.75, 32.6	1520.85, 30.7	1675, 49.9	1573.6, 1.2
		3	1325.6, 32.1	1313.6, 30.6	1403.2, 51.6	1379.4, 1.2
		4	1494.85, 34.5	1489.05, 32.4	1571.6, 55.1	1541.2, 1.3
		5	1294.25, 32.8	1293.73, 29.9	1376.8, 50.7	1310.8, 1.5

. . . continues

Table 3 (continued)

Problem size	α	Data set	Meta-heuristic	Modification 2	Generic	Heuristic
	1.0	1		20961.5, 37.5	24413, 59	21620, 2.8
		2		20685.25, 41.0	23809, 59.9	21681, 3.2
		3		20251.75, 32.0	23176, 58.9	20979, 3.5
		4		22545.2, 36.2	25549, 58.8	23316, 3.2
		5		19501, 43.0	21795, 59.5	20361, 3.2
	0.5	1		10746, 37.5	12202.5, 59	11074.5, 2.6
100		2		10637.25, 31.9	12110.5, 61	11020.5, 2.5
		3		10357, 32.8	12022, 58.9	10745.4, 2.8
		4		11689.88, 32.8	12789.5, 60	11942, 2.7
		5		9902.2, 35.6	11072, 60.6	10425.5, 3.0
	0.2	1		4583.35, 32.4	5180.4, 57.0	4744, 2.3
		2		3536.35, 34.2	5116, 57.4	4755.4, 2.2
		3		4437.4, 34.6	5160.6, 58.6	4605, 2.2
		4		4876.45, 33.4	5523.2, 58.8	5113.4, 2.2
		5		4247.9, 34.2	4749.2, 57.7	4464.2, 2.4

It can be seen from tables 1 and 2 that optimal or near-optimal solutions are obtained for most of these problems. The methods employing the branch-and-bound algorithm, namely the meta-heuristic and modification 1, provide good solutions for most of these problems. This is expected since the branch-and-bound provides a large amount of fathoming in most of these cases. As a result, the solution quality should improve when the schedule disruption probabilities are increased. Comparing the population averages of the OFVs over 200 generations for the 14-job problems (figure 2), it is clear that the objective (a minimization function), decreases as the disruption probabilities are increased. This characteristic was observed for the 10-job problem also. For smaller problems, the heuristic solutions are also close to the optimal in many cases. The advantage of using the meta-heuristic and its modifications over the branch-and-bound is apparent when CPU times and solution quality are considered. The CPU times for the former are consistently low for all the data sets. However, for the branch-and-bound algorithm there is a large variance in CPU times over different data sets. Some problems have consumed more than 5000 seconds. In general, none of the heuristics developed in this paper dominate over each other. However, the good solutions provided by the meta-heuristic and modification 1 do indicate the usefulness of augmenting the search of generic GA with problem-specific information. The lower bounds for all problems are shown in parentheses in the optimal OFV column. The gap between lower bounds and optimal OFVs tends to

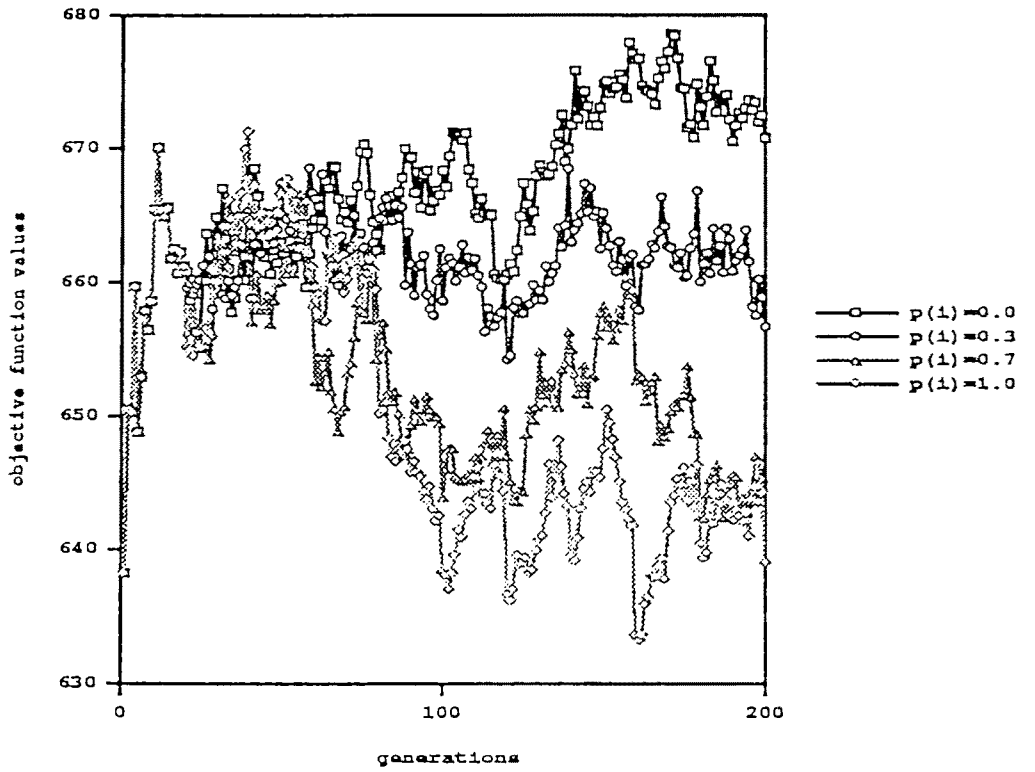

Figure 2. Meta-heuristic population averages over generations for a 14-job problem.

become worse when the input data is such that there is much idle time in the schedules. As a result, they cannot be used as a good benchmark for determining the quality of solutions. Hence, for larger problems in table 3, we provide the OFVs of solutions generated by the heuristic algorithm and generic GA.

Table 3 shows results of our experimentation with medium sized and large problems – problems with 20, 30, 50 and 100 jobs. Five randomly generated data sets are used for each of these problems. Since optimal solutions are not known and in the absence of a better benchmark, the algorithms developed in this paper are compared with generic GA and the heuristic discussed in section 3. Also, because modification 1 performs as well as the meta-heuristic for smaller problems, it was excluded from our experimentation with the larger problems. Thus, only the meta-heuristic and modification 2 are included in table 3. This allows us to focus on the role of information provided by the branch-and-bound with increasing problem size.

From table 3, we notice that modification 2 and the meta-heuristic outperform the generic GA and the heuristic. It shows that by providing problem-specific information via a branch-and-bound algorithm to GA, its performance can be significantly enhanced. Note that the generic GA provides worse solutions than the meta-heuristic or modification 2 and requires more CPU time. The heuristic provides worse solutions

but requires less CPU time. Also, between the meta-heuristic and modification 2, the latter outperforms the former in a number of cases, whereas, in other, the meta-heuristic provides better solutions. Again, neither of these two algorithms can be declared a winner. The usefulness of modification 2 over the meta-heuristic becomes more apparent as problem size is increased. The latter requires relatively more CPU time and because there is insufficient node fathoming information, it is not able to provide as good solutions as modification 2. This is a disadvantage of the meta-heuristic. For example, when the problem size is increased to 100, CPU time incurred by the branch-and-bound algorithm (and hence the meta-heuristic) increases considerably. Hence, for the 100-job problem, we limit our experimentation to modification 2. Again, comparing modification 2 with the heuristic and GA algorithms indicates that directing the search of GA in areas where optimal/near-optimal solutions are likely to exist improves the performance of pure GA.

5. Summary and conclusions

In this paper, we presented a GA based meta-heuristic and its modified versions for solving a flowshop scheduling problem. The criteria included were average job flowtimes and makespan. Computational results were provided for 10-, 14-, 20-, 30-, 50- and 100-job problems. From the results, it is clear that the information provided by branch-and-bound plays an important role in the good performance of the meta-heuristic and modification 1. When CPU times and solution quality are considered, both algorithms outperform pure GA and pure branch-and-bound. They work well when branch-and-bound provides sufficient information and appear to be suitable for medium sized problems with up to 50 jobs. With an increase in problem size, the problem-specific information available via branch-and-bound is limited and hence the meta-heuristic and modification 1 are not suitable. In such instances, modification 2 provides good solutions and requires low CPU times.

The meta-heuristic and its modifications are shown to be efficient since their computational requirements do not increase exponentially with problem size. Thus, they are good candidates for solving problems which are computationally hard for conventional optimization procedures.

References

[1] W.J. Selen and D. Hott, A mixed integer goal programming formulation of the standard flow-shop scheduling problem, J. Oper. Res. Soc. 37(1986)1121.

[2] J.M. Wilson, Alternative formulation of a flowshop scheduling problem, J. Oper. Res. Soc. 40(1989)395.

[3] C. Rajendran, Two-stage flowshop scheduling problem with bicriteria, J. Oper. Res. Soc. 43(1992)871.

[4] A. Nagar, J. Haddock and S.S. Heragu, Multiple and bicriteria scheduling: A literature survey, Euro. J. Oper. Res. 81(1995)88.

[5] J.H. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, 1975).

[6] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison–Wesley, 1989).

[7] G. Cleveland and S. Smith, Using genetic algorithms to schedule flow shop releases, in: *Proc. 2nd Int. Conf. on Genetic Algorithms* (1987) p. 160.
[8] M. Husbands and F. Mills, Simulated co-evolution as the mechanism of emergent planning and scheduling, in: *Proc. 4th Int. Conf. on Genetic Algorithms* (1991) p. 264.
[9] R. Nakano and T. Yamada, Conventional genetic algorithm for job shop problem, in: *Proc. 4th Int. Conf. on Genetic Algorithms* (1991) p. 474.
[10] G. Syswerda, Schedule optimization using genetic algorithms, in: *Handbook of Genetic Algorithms* (Van Nostrand Reinhold, 1991) p. 333.
[11] S. Bagchi, S. Uckun, Y. Miyaben and K. Kawamura, Exploring problem-specific recombination operators for job-shop scheduling, in: *Proc. 4th Int. Conf. on Genetic Algorithms* (1991) p. 10.
[12] B.R. Fox and M.B. McMahon, Genetic operators for scheduling problems, *Genetic Algorithms and Simulated Annealing* (1993) p. 284.
[13] D. Whitley, T. Starkweather and D. Shanner, The traveling salesman and sequencing problem: Quality solution using genetic edge recombination, in: *Handbook of Genetic Algorithms* (Van Nostrand Reinhold, 1991) p. 350.
[14] G. Syswerda and J. Palmucci, The application of genetic algorithms to resource scheduling, in: *Proc. 4th Int. Conf. on Genetic Algorithms* (1991) p. 502.
[15] A. Nagar, S.S. Heragu and J. Haddock, A branch and bound approach for a two-machine flowshop scheduling problem, J. Oper. Res. Soc. 46(1995)721.
[16] P. De, J.B. Ghosh and C.E. Wells, Heuristic estimation of the efficient frontier for a bicriteria scheduling problem, Dec. Sci. 23(1992)596.