TACKLING COMPLEX JOB SHOP PROBLEMS USING OPERATION BASED SCHEDULING

D S Todd and P Sen
Engineering Design Centre and Dept. of Marine Technology
University of Newcastle Upon Tyne, NE1 7RU, UK.
email: d.s.todd@ncl.ac.uk, pratyush.sen@ncl.ac.uk

Abstract. Scheduling is a combinatorial problem with important impact on both industry and commerce. If it is performed well it yields time and efficiency benefits and hence reduces costs. Genetic Algorithms have been applied to solve several types of scheduling problems; Flow Shop, Resource, Staff and Line Balancing have all been tackled. However Job-shop Scheduling is the most common problem of interest. The Job-shop Scheduling Problem (JSP) involves placing jobs onto a set of machines with the aim of minimising makespan, the total time to complete all jobs. The standard JSP model is relatively simple and cannot cope with many real world situations. This paper outlines a new method, using an operation based construction, which expands the JSP to increase its functionality and applicability. This allows for much more complex job shop problems involving flexible machines, setup and maintenance times, deadlines and multiple optimisation criteria. The method is fully explained and demonstrated using a 20 job-4 machine example.

1. Introduction

The Job-shop Scheduling Problem (JSP), first attempted with GAs by Davis [1], is one of the hardest [2] and most commonly encountered scheduling problems. Since Davis' work there have been many attempts at the simple JSP with a single criterion [3-9]. More complex approaches by Liang [10] and Sridhar [11] used a weighted sum approach to combine disparate metrics to give a single fitness measure. Murata [12] used a random weight system to generate a range of trade-off solutions in a flowshop problem, while Hamada [13] addressed optimization with constraints. However, none of these approaches combine multiple criteria considerations and the increased job shop complexity which is considered in this paper.

A classical JSP is defined as follows:

i) There are k machines and n different *jobs*.
ii) Each job is composed of a set of m or less *stages*.
iii) Each stage requires a specific machine.
iv) Each stage has a fixed processing time.
v) The stages must be carried out in a specific order.

vi) There are no precedences between different jobs.
vii) Operations cannot be interrupted.
viii) Each machine can process only one job at a time.
ix) Neither release times nor due dates are specified.

The aim is to determine the operation sequences on the machines in order to minimise the makespan. To demonstrate this problem a 4 job 3 machine example is used (Table 1.1). M1 denotes that machine 1 is needed to perform the operation and the following number indicates the time required to perform it. The ordering of the jobs on each machine is very significant. This is demonstrated in Figure 1.1. The first case is randomly generated and has a makespan of 38. The effect of optimisation can be clearly seen in the second example. The makespan has been reduced to 31. This effect becomes more significant as problem size increases. The final example shows what can be achieved if the flexibility of the machines is increased, in that the same stage of a job can be performed on more than one machine, an extension which is discussed later. In this case machine 2 has been modified to allow it to perform machine 1's jobs as well as its own. This reduces the makespan to 24.

	Stage 0	Stage 1	Stage 2	Stage 3	Stage 4
Job 0	M1 ,5	M0, 3	M1, 4		
Job 1	M2, 3	M1, 6	M0, 5	M1, 3	M2, 4
Job 2	M0, 4	M2, 3	M1, 6		
Job 3	M1, 4	M2, 5	M1, 3	M0, 5	

Table 1.1 - Job Shop Problem

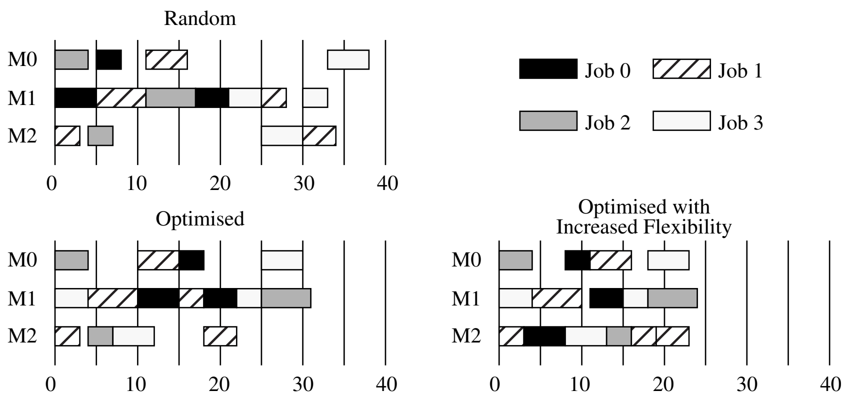

Figure 1.1 - The Effect of Ordering

2. Complex Scheduling

Scheduling plays an important role in most manufacturing environments. It can lead to reduced lead times and inventory levels as well as improved integration between different functions within an organisation such as manufacturing and assembly . This increasing level of interaction facilitates the use of Concurrent Engineering techniques to increase process efficiency. However integration requires a systematic consideration of manufacturing to deadlines. Deadlines will occur internally but often a company will have to pay compensation for every day or week a product is delivered late to the customer.

It is clear from the above description that the standard JSP does not realistically represent these problems and hence is very limited in its application. At this point a specification for the Complex JSP was developed to allow for these additional requirements.

The Complex JSP implements the following additions to the Simple JSP:

i) Each stage requires a specific *operation*.
ii) Each stage has a *standard processing time*.
iii) A job can visit the same machine one or more times.
iv) There are interdependencies between different jobs.
v) Release times and due dates are specified.
vi) Machines can perform several operations.
vii) Machines perform operations with different efficiencies.
viii) Machine maintenance times can be specified.
ix) Machine setup times for operations can be specified.
x) Due date penalties can be specified.
xi) Multiple scheduling criteria can be optimised.

The Complex JSP model has a much greater level of complexity and range of possible applicability. There are still n-jobs to be tackled. Each of these jobs has m stages but instead of specifying a specific machine an operation is specified for each stage. The motivation for operation based scheduling is outlined more fully in section 3. Each stage is also given a standard processing time which is the time it takes to perform that operation on a 'standard' machine. The stages must be processed in a specific order as before but a stage can involve going on a specific machine more than once, consecutively if necessary. Jobs are not given precedences but interdependencies can be specified. For example if a job requires the completion of other jobs before it can start this can be specified within the problem. Operations cannot be interrupted unless it is for maintenance. It is possible to specify whether a job should wait until after maintenance. The jobs also have release times and due dates. The release times allow scheduling to be integrated with previous events such as the completion of a design task or the delivery of materials which prevent a job starting. Due dates can also be specified and the severity of penalties on these due dates can also be set. Hard due dates can be given large penalties but ones where slippage is allowed can be less heavily penalised. Machines can be flexible and are able to perform more than one type of operation. The time it takes the machine to process a given stage is dependent on its efficiency at performing the operation required at that stage. With the machines performing different operations machine setup times must also be considered within the schedule plans. Finally, scheduling is not just about minimising makespan, in complex scheduling it also involves the consideration of several conflicting criteria. This is discussed further in section 4.

3. Operation Based Scheduling

Schedulers have commonly used the classical job shop approach which assumes that any given stage of a job can be undertaken at only one machine and the scheduling task is to order the handling of the stages on the given machines. This is often not the case in practice as a workshop may have several machines of the same type or various machines which can carry out the same operations. Therefore the operations needed for any one stage of a job could be carried out on a number of different machines. There are also other complications involved in this type of model. If a machine is set up for one type of operation and then switched to another type of operation there is some setting up involved. Additionally different types of machines performing the same operation may also take different amounts of time (e.g newer machines will operate faster than older ones). These factors indicate that the scheduler required must be capable of handling operations and not jobs.

4. Multiple Criteria Scheduling

Scheduling has commonly been tackled using Linear Programming, Monte-Carlo and Heuristic methods [14]. The aim of such procedures has been to minimise makespan, the total time from the start of the first task to the end of the last. However, other important factors exist within a scheduling task. Regulated allocation of resources, limits on the allowed number of tasks in progress, the completion of tasks by specific deadlines, interrelationships between tasks and conflicts for shared resources are just a few examples. Solutions are sought which strike a balance between these potentially competing requirements.

As a simple example a workshop with limited space may be considered. One objective is to limit the amount of work in progress, the other is to minimise the time taken to complete a set of tasks. The trade-off can be clearly shown on a trade-off curve (Figure 4.1). We can either minimise makespan (1) or minimise the maximum level of work in progress (3). However a compromise solution (2) may be more preferable. The solutions to the problem will be confined within a feasible region which is defined by system limitations and imposed constraints. Hence, the aim of scheduling in a multi-criteria context is to generate the trade-off boundary. To do this a Multiple Criteria GA (MCGA) may be used. This is explained elsewhere [15]. The generated boundary can then be used to choose the solution that has the most desirable balance of criteria for the given set of circumstances. The most desirable balance is obviously dictated by the priorities in question.

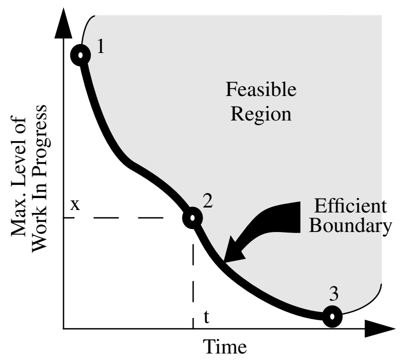

Figure 4.1 - Trade-off in Scheduling

5. Multiple Criteria Scheduler

5.1. Complex Job Shop Scheduling

The complexity of the scheduling problem can no longer be represented by a single table showing jobs, stages, machines and times as in Table 1.1. Several tables relating to different aspects of system are needed to fully specify the problem and these are explained below. The system uses the user defined clock tick as a time unit. The clock tick can be of any real-time duration (1 minute, 10 minutes, 1 hour, etc.). However, the higher the definition (smaller the clock time) the longer the scheduler will take to run for a given length schedule as there will necessarily be more clock cycles.

The problem in question is defined in terms of several variables: Number of Machines, Number of Operations, Number of Jobs, Maximum Number of Stages, Maximum Number of Dependencies and Maximum Number of Maintenance Periods. It also has a variable which indicates whether job splitting is allowed. Job Splitting occurs when a job is on a machine and scheduled maintenance has to take place before the job is complete. This results in the job being stopped and restarted after maintenance. If job splitting is not allowed the scheduled job will not be started until after maintenance has ceased.

The first table used is the machine operating efficiency table (Table 5.1). This is used to show the efficiency of machines when performing the different operations. Those with an efficiency of 1.0 perform operations in the standard specified time. Those with an efficiency greater than 1 perform it faster whilst those with less perform it slower. Machines with efficiency 0 cannot perform the operation. For example machine 0 can perform Op0 with 100% efficiency and Op3 with 50% efficiency but cannot perform any other operations.

	Op0	Op1	Op2	Op3	Op4	Op5
M0	1.0	0	0	0.5	0	0
M1	0	0.5	0	0	0.8	0
M2	0	1.0	0.8	0	0	0.8
M3	0.5	0.8	0	0.3	0	1.0

Table 5.1 - Machine Operating Efficiency Table

	No.	N/A 0	N/A 1	N/A 2	N/A 3
M0	2	50-60	100-110		
M1	1	20-25			
M2	4	10-15	25-30	34-37	50-52
M3	2	20-35	60-64		

Table 5.2 - Machine Maintenance Times

The machine maintenance times are shown in Table 5.2. If the machine requires new parts to be fitted or to be stripped down and cleaned then routine maintenance times can be allocated. For each machine the number of periods is noted and for each of these periods a time range is specified. For example, machine 3 has 2 periods where it is not available, 20-35 clock ticks and 60-64 clock ticks

If a machine is able to perform several different types of operation the transition between operation types may require some setup time. These times are specified in Table 5.3. The current operation is on the left and the next operation reads down from the top. If a transition from Op3 to Op4 takes place 2 clock units will be required to complete this transition. Inclusion of this facility allows the MCGA to experiment with performing all of one type of operation on one machine, hence no setup time would be needed, or changing machine operation types. The effects of this can only be found through simulation, since it is case dependent.

In Table 5.4 each job has a number of stages each of which is specified in terms of two numbers. The first is the operation type which has to be performed at that stage. The second is the standard time required at that stage.

	Op0	Op1	Op2	Op3	Op4	Op5
Op0	0	4	2	2	3	1
Op1	3	0	1	5	4	2
Op2	3	2	0	2	3	1
Op3	4	3	2	0	2	1
Op4	3	5	2	1	0	2
Op5	5	4	4	1	2	0

Table 5.3 - Operation Setup Times

	No.	S0	S1	S2	S3	S4
J0	2	2,4	4,10			
J1	4	1,6	2,3	5,8	0,3	
J2	3	4,5	1,3	2,8		
J3	4	0,7	2,7	4,3	5,1	
J4	5	1,4	0,5	1,7	3,9	4,6
....	....	....	....	....	....	....

Table 5.4 - Job-Stage Table

If Table 5.4 and Table 5.1 are brought together the effect of efficiency can be demonstrated. For an example let job 3-stage 0 be chosen from Table 5.4. This stage requires operation 0 to be performed. Using Table 5.1 it can be seen that machine 0 and machine 3 can perform this operation. If machine 3 is chosen this machine performs this operation with efficiency 0.5. The total time taken for this operation is given by:

$$\text{Time} = \frac{\text{Standard Time}}{\text{Efficiency}}$$

Hence, on machine 3, job 3-stage 0 will take 7/0.5 = 14 clock ticks.

Similarly, job 3-stage 0 will only take 7 clock ticks on machine 1, because the efficiency is 1. If the result of this calculation is a decimal it is rounded up to the nearest clock tick in this implementation. This means that over estimates will be provided. If a shorter clock tick unit is used this error becomes smaller but as mentioned earlier the simulation will take longer.

The final table (Table 5.5) used specifies the earliest start times, dependencies and deadlines for each job. The column labelled ‘start’ shows the earliest time that job can commence. This may be due to the delivery date of raw material or the date for completion of a part design. The next column specifies the number of dependencies that this job has. For example job 3 has 2 dependencies, and these relate to jobs 7 and 8 (from the following columns). This means that before job 3 can start both job 7 and job 8 must be completed. Finally, the last two columns specify the deadlines and penalties for the jobs. The deadline is the clock time at which the job is due. After this point penalty costs are incurred. Job 3 has a stricter deadline than the other jobs because it has a 5 unit cost penalty. Job 4 incurs no penalties whatever time it finishes.

	Start	No.	D0	D1	D2	Deadline	Penalty
J0	10	3	6	8	9	100	1
J1	0	0				50	1
J2	5	0				90	1
J3	0	2	7	8		100	5
J4	0	1	16			0	0
....	....	....	....	....	....	....	....

Table 5.5 - Job-Stage Table

5.2. String Configuration in MCGA

Due to the nature of this scheduling problem it is not possible to use a string which is partitioned into machine sections each containing a prioritised, fixed length, list of stages. This is because the possible interchange of stages between machines would mean that although strings would remain the same length overall, the boundaries between machine partitions would change as genes move around in the string. This could also cause the GA to introduce duplicates during crossover. The system used here still partitions the string into machines but it holds multiple copies of stages which occur on several machines. By using a group of activation genes inserted at the start of the string it activates/disables the duplicated stages within the string. This technique was developed from the ideas of the Structured GA [16].

5.3. Selective Gene Activation

The structured GA techniques are based on a binary string. However if the concept is extended to use an integer string a different type of activation can be initiated. The string is the same as before but instead of being able to turn on and off sets of genes the upper level genes can now choose which genes they activate (Figure 5.1).

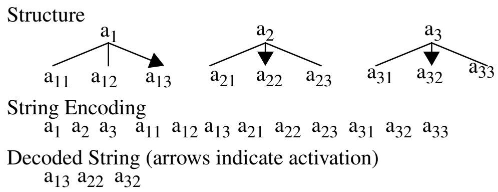

Figure 5.1 - Gene Activation

5.4. String Construction

In the context of the scheduler, selective gene activation can be used to choose which jobs are performed on which machines. This technique is demonstrated here using a simple example. Table 5.6 is a table of the machine operating efficiencies and Table 5.7 shows the Job-Stage data. Each stage of a job is now defined in terms of the operation that needs to be performed at that stage. Operations can be thought of as specific tasks (e.g drilling or milling in a machine shop) and it is usually the case that more than one machine can perform this operation. Operations can still be restricted to a single machine (Op0, Op1, Op2 in this example) so the simple JSP can still be tackled. The data can be interpreted as follows: Job 0-Stage 1 involves Op4 which can be performed on either Machine 0 or Machine 2.

	Op0	Op1	Op2	Op3	Op4	Op5
M0	1.0	0	0	1.0	1.0	0
M1	0	1.0	0	1.0	0	1.0
M2	0	0	1.0	0	1.0	1.0

Table 5.6 - Machine Operating Efficiency Table

	No.	S0	S1	S2	S3	S4
J0	3	1,5	4,3	1,4		
J1	5	2,3	1,6	3,5	1,3	4,4
J2	3	0,4	2,3	1,6		
J3	4	5,4	2,5	1,3	0,5	

Table 5.7 - Job-Stage Table

The string is made up of n+1 distinct segments, where n is the number of machines and the extra segment is made up of a set of activation genes. Each of these activation genes will correspond to one of the machine choices (i.e. stages which can be performed by more than one machine) so in this case there will be 4 activation genes. This is because only operation types 3,4 and 5 can be performed on more than one machine, the other operations being limited to a single machine, as Table 5.6 shows. In each of the machine segments there is a gene for each of the stages which are to be performed on that machine, including those which could appear on two or more machines. Hence the string is a concatenation of these segments:

Machine Choices - Machine 0 - Machine 1 - ..... - Machine i - ..... - Machine n

In the case of the example we have:

Machine Choices = j0s1, j1s2, j1s4, j3s0
Machine 0 = j0s1, j1s2, j1s4,j2s0, j3s3
Machine 1 = j0s0, j0s2, j1s1, j1s2, j1s3, j2s2, j3s0, j3s2
Machine 2 = j0s1, j1s0, j1s4, j2s1, j3s0, j3s1

To translate this into a string which can be manipulated by the MCGA several encoding steps take place. First the machine choice section is constructed by concatenating the machine choices in ascending job-stage order. In this example there are four machine choices: j0s1, j1s2, j1s4, j3s0. This means that there are four integer activation genes in the machine choice section. Taking the first gene, j0s1, this is of type Op4. Looking in Table 5.6 this can go either on machine 0 or on machine 2. Hence the value of the gene will be restricted to either 0 or 2. For j1s2, this can go on machines 0 or 1, hence this gene will be restricted to either 0 or 1 and so on for the other machine choices. Following this each machine section is constructed. These sections are similar to the Travelling Salesman Problem in construction. Taking Machine 0 as an example there are 5 job stages so this string section will be 5 genes long. Each job is simply represented by its job number and its stage number is not used. For this example a sample string would be 0,1,1,2,3. Any permutation of these five genes is legal; for example 1,2,0,1,3 or 3,1,2,1,0. The order of the jobs form a preference list for that machine with the first job being attempted given highest priority. The stage numbers do not need to be included due to the implied ordering of jobs. In the case of machine 1 there are two 1's, but in order for j1s4 to be started j1s2 must already have been completed, hence stage 2 must come first. This encoding is repeated for all machines until the string is complete. The initial process of constructing the string is done automatically from the input datafile. The user does not need to specify machine choice restrictions, segment lengths and job contents or machine sections.

As the system uses an indirect notation a schedule builder is used. This was done to allow the extra functionality required. The schedule builder requires a preference list for each machine in order to carry out processing. Within these lists each job stage should only appear once.

To demonstrate the decoding process a sample string from the above example is decoded. The string is first separated into machine choice and machine sections.

Machine Choice	Machine 0	Machine 1	Machine 2
0, 0, 0, 2	3, 1, 2, 1, 0	2, 1, 1, 0, 3, 3, 0, 1	3, 1, 0, 1, 3, 2

The machine choice string is now used to activate the corresponding jobs in each of the machine lists. In all other lists these jobs will be removed. For example, the first gene in the machine choice list is for job 0 stage 1. This job can either go on machine 0 or machine 2. In this case machine 0 has been selected hence job 0 stage 1 must be removed from machine 2. Looking at the Machine 2 there is only one job 0 so this is removed. If there is more than one stage of the same job on the machine the implicit job ordering is used. For example, the second machine choice gene belongs to job 1 stage 2. Machine 0 has been chosen so the other machine location for this job, machine 1, has to be removed. It can be seen that machine 1 has three job 1 genes. By referring to Table 5.6 and 5.7, job 1 stage 2 is the second possible job 1 stage to appear on machine 1, the first being job1 stage 1. Hence the second 1 is removed from the machine 1 list. If this is continued for all machine choice sections the following machine preference lists result:

Machine 0 Preference List:	3, 1, 2, 1, 0
Machine 1 Preference List:	2, 1, 0, 3, 0, 1
Machine 2 Preference List:	3, 1, 3, 2

All the extra jobs have been removed from the machine sections and no duplicates exist. Using these lists the schedule builder can now be invoked to build a schedule and collect statistics on the data.

5.5 Crossover and Mutation Operators

A complete string operator cannot be used on the string because it is separated into several distinct sections. Instead crossover is applied to each of the individual sections in the undecoded string. The machine choice section is crossed using a simple two point crossover. This means that two random points are selected within the machine choice string section and the genes between these points are swapped between partners. This is valid as this section is not order based.

Within each of the machine job list sections a new operator is employed based on a modification of the enhanced edge crossover operator [17]. In these sections of the string the two-point crossover operator cannot be used because it would introduce repeated jobs. The modified operator is slightly more complex, but it ensures that repeated jobs are not created and jobs are not lost during crossover. The procedure is shown in Figure 5.2.

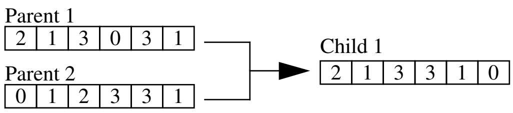

Figure 5.2 - Crossover in Machine Sections

Using parent 1 the process is started from job 2. Then the following jobs in both parents, 1 and 3, are found and a choice is made between them at random. If 1 is chosen the next jobs are 3 and 2. Job 2 only appears once and has already been used so job 3 must be used. Following job 3 in the parent strings are jobs 0 and 3, and 3 is chosen at random. Following the second job 3 is job 1 in both cases so this is chosen. Job 0 is the first job in parent 1 unused so is inserted at the end. This process is repeated from the first job in parent 2 to create another child. This operator tries to consider the immediately following job and also the overall ordering by trying to select the earliest jobs in cases of conflict. If two identical strings are present the children will be the same as the parents.

Mutation is a much simpler operator. Again mutation is performed on separated sections of the string. In the machine choice section, if a gene is chosen it is randomly reset to another machine choice. For example if there is a choice of three machines 0, 2 and 3 for a specific job and the current machine selected is 2 it would change to either 0 or 3. In the machine job list sections a gene selected for crossover will simply be swapped with another randomly selected gene in that section.

Due to the combinatorial complexity of the problem a small local search is initiated which behaves like an intelligent mutation operator. This search chooses a first job and then a second is picked randomly and the two are swapped. If this swap provides improvement in at least one of the scheduling criteria without loss in any of the others then the swap is accepted. If this is not the case the jobs are swapped back and another job is picked randomly to swap with the first. This process is repeated up to 5 times if a good swap is not found then the string is left untouched. These swaps occur within each machine section.

5.6. Schedule Builder

The schedule builder takes the strings provided by the MCGA and decodes them into machine preference lists as outlined before. These lists are then used to create the schedule. The builder will always create the same schedule for a given input string. The builder starts by resetting the clock to zero. Then the system enters a loop which is terminated when all jobs have finished. This loop starts by looking at each machine to determine its current state, either setup, maintenance, working or idle. If the machine has finished a job, the job is marked complete, its finishing time is noted and the machine set to idle. If the machine is either working, in setup or in maintenance the time to completion of this task is decremented by 1 clock tick. If the machine is working, then the time of utilisation of that machine is incremented.

Following this the preference list of each machine which is idle is examined to find the highest priority job which is available and has not yet been performed. Availability is defined in terms of several factors. A job is available if:

i) All previous stages of that job are complete.
ii) All dependencies of that job are satisfied.
iii) The machine is set up for that job.
iv) If job splitting is not permitted the job must be completed before any maintenance begins.

In the cases of i), ii) and iv) the next job in the preference list of the machine is examined to see if it is available. If iii) is the case the machine begins its setup routine for the operation of that job. The time required for setup is determined by the current machine operation and the operation required by the new job and is found in the operation setup time table (Table 5.3). If no job on the list is available the machine is left in its idle state. If a job is found then the time required to complete that job is calculated and it is placed on the machine. The clock is then incremented and the process returns to the start of the loop.

When all jobs are complete the code exits the loop and various statistics can be calculated such as makespan, penalty costs, average job times, machine utilisations etc. These are passed back to the MCGA for analysis. The code will also generate plans showing each machine, what jobs and stages are being performed, any setups, maintenance times, idle times and also statistics from the run.

6. Scheduler Testing

The scheduler was tested to check for correct operation by examining the results of the MCGA. In this example there are 4 machines, 20 jobs with a maximum of 5 stages and 6 types of operations. It includes all of the complex features discussed earlier. The search space contains at most $3^{12}x2^{23}x16!x20!x35!x35!$ possible combinations. This is calculated by finding the number of machine choices, there are 12 stages with 3 choices (3^{12}) and 23 stages with 2 choices (2^{23}). This is then multiplied by the number of possible combinations in each of the machine sections (16!, 20!, 35!, 35!). The search space will actually be smaller than this due to the fact that some of the jobs are repeated in the machine sections. This is the same as having the same city occurring twice or more times in the Travelling Salesman Problem. This test was run using two criteria, makespan and average job time, the latter being defined as the sum of time taken by each job divided by the number of jobs. The aim of the average job time criterion is to reduce the number of jobs which are in progress at one time. This criterion may be used in small workshops where storage space is at a premium.

For the tests a population of 500 was run over 100 generations. A typical run on this example takes about 25 minutes. The graphs showing the development of the trade-off solutions over the 100 generations are shown in Figure 6.1. The effect of job splitting was also tested. Figure 6.2 shows the final population of two runs, with and without job splitting.

Firstly the schedule plans generated by the system were examined for validity. The job list was checked to ensure all jobs were completed. The timings of the jobs were checked as were the setups when required. The MCGA Scheduler generates an average of 72 solutions per test over a 20 test run. It was found that multiple solutions would generate the same trade-off values even though they were genotypically different. This is why the graphs do not appear to have all of the 72 solutions present. The schedule plans reveal that, in general, in order to reduce makespan the average job time would increase. This is because the scheduler has to be more efficient with resources, so job stages need to be moved around so that they fit into gaps within the schedule. To minimise average job times, the schedules which group several job stages together were favoured. This leads to large gaps in the schedule where jobs will not commence until necessary resources are free. In the extreme case minimising average job time is equivalent to a 'no-wait' job shop where no gaps are allowed between job stages.

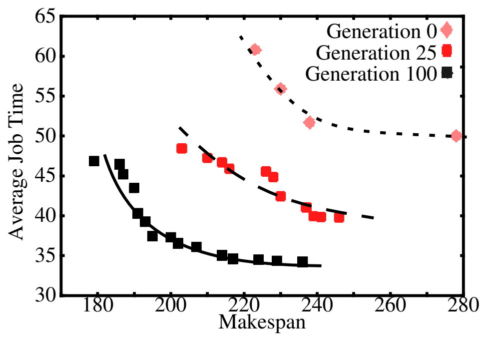

Figure 6.1- Generation of Trade-off Solutions

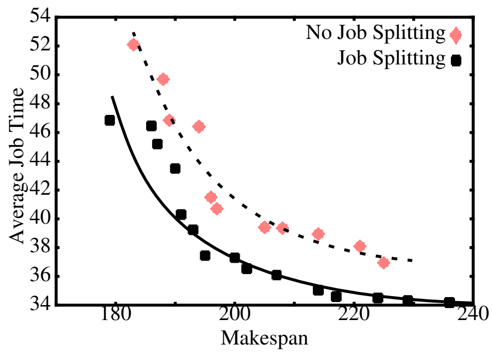

Figure 6.2 - Job Splitting vs No Job Splitting

As mentioned earlier Figure 6.2 shows the effect of job splitting. Clearly using job splitting has a positive effect on both makespan and Average Job Time. If job splitting is not permitted then jobs are held until such time as they can be completed without any interruption. However if there are short jobs which can be completed these will be selected to fill in the gaps. The problem arises when there are long jobs and short gaps between maintenance periods. The jobs are continually put back until large enough windows exist. This sometimes leads to long delays especially on machines with lots of scheduled maintenance. In this case the best schedules are those where as many jobs as possible are placed onto other machines effectively bypassing the troublesome machine. The current algorithm does not allow the jobs to be removed from machines by other jobs of higher priority. This is a possible extension for future work.

7. Conclusions

This paper has outlined the use of operation based scheduling and evolutionary techniques to solve complex scheduling problems. The simple job shop model was thought not to be detailed enough for application to many scheduling problems. For this reason a complex scheduling definition was drawn up incorporating many new features, such as the use of multi-operation machines, job dependencies and setup periods.

Following this a description of operation based scheduling along with the data required to perform it is given. The construction of the MCGA Scheduler is outlined and the string representation, crossover and mutation operators and the schedule builder are also presented.

The MCGA Scheduler is found to be a useful and flexible optimisation tool providing insights into the processes involved in complex scheduling. Further investigations are being carried out on more real world examples with the aim of improving efficiency and the effectiveness of genetic operators. Additionally, system functionality continues to be enhanced in order to increase the range of problems which the scheduler can tackle.

8. References

1. Davis L, 1985. Job Shop Scheduling with Genetic Algorithms. *Proceedings of The First International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Hillsdale, NJ, pp136-140.
2. Tsujimura Y, Cheng R, Gen M, 1997. Improved Genetic Algorithms for Job-Shop Scheduling Problems. *Engineering Design and Automation* 3(2):133-144.
3. Nakano R, Yamada T, 1991. Conventional Genetic Algorithm for Job Shop Problems. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, pp474-479.
4. Syswerda G, Palmucci J, 1991. The Application of Genetic Algorithms to Resource Scheduling. *Proceedings of the Forth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, pp502-508.
5. Bruns R, 1993. Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, pp352-359.
6. Fang H-L, Ross P, Corne D, 1993. A Promising Genetic Algorithm Approach To Job-Shop Scheduling, Re-Scheduling and Open-Shop Scheduling Problems. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, pp375-382.
7. Gen M, Tsujimura Y, Kubota E, 1994. Solving Job Shop Scheduling Problems by Genetic Algorithm. *IEEE International Conference on Systems, Man and Cybernetics, Vol.2.* pp1577-1582.
8. Reeves C R, 1995. A Genetic Algorithm for Flowshop Scheduling. *Computers in Operations Research*, 22(1):5-13.
9. Cheng R, Gen M, Tsujimura Y, 1996. A Tutorial Survey of Job-Shop Scheduling using Genetic Algorithms - I.Representation. *Computers and Industrial Engineering* 30(4):983-997.
10. Liang S J, Lewis J M, 1994. Job Shop Scheduling Using Multiple Criteria. *Proceedings of the Joint Hungarian-British Mechatronic Conference*. pp77-82.
11. Sridhar J, Rajebdran C, 1996. Scheduling in Flowshop and Cellular Manufacturing Systems with Multiple Objectives - A Genetic Algorithmic Approach. *Planning Production and Control* 7(4):374-382.
12. Murata T, Ishibuchi H, Tanaka H, 1996. Multi-Objective Genetic Algorithm and it's Application to Flowshop Scheduling. *Computers in Industrial Engineering* 30(4):957-968.
13. Hamada K, Baba T, Sato K, Yufu M, 1995. Hybridizing a Genetic Algorithm with Rule-Based Reasoning For Production Planning. *IEEE Expert* October:60-67.
14. Muth J F, Thompson G L, 1963. *Industrial Scheduling*. Prentice Hall Inc., Englewood Cliffs, New Jersey.
15. Todd D S, Sen P, 1997. A Multiple Criteria Genetic Algorithm for Containership Loading. *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, ISBN 1-55860-487-1, pp674-681.
16. Dasgupta D, McGregor D, 1992. Structured Genetic Algorithms: The Model and First Results. Department of Computer Science, University of Strathclyde, UK, 1992.
17. Starkweather T, McDaniel S, Mathias K, Whitley D, Whitley C, 1991. A Comparison of Genetic Sequencing Operators. *Proceedings of The Third International Conference on Genetic Algorithms*. Morgan Kaufmann, ISBN 1-55860-308-9, pp69-76.