J Intell Manuf (2012) 23:1063–1078
DOI 10.1007/s10845-011-0520-x

A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study

S. Meeran · M. S. Morshed

Received: 17 June 2010 / Accepted: 11 February 2011 / Published online: 8 March 2011

Abstract In recent decades many attempts have been made at the solution of Job Shop Scheduling Problem using a varied range of tools and techniques such as Branch and Bound at one end of the spectrum and Heuristics at the other end. However, the literature reviews suggest that none of these techniques are sufficient on their own to solve this stubborn NP-hard problem. Hence, it is postulated that a suitable solution method will have to exploit the key features of several strategies. We present here one such solution method incorporating Genetic Algorithm and Tabu Search. The rationale behind using such a hybrid method as in the case of other systems which use GA and TS is to combine the diversified global search and intensified local search capabilities of GA and TS respectively. The hybrid model proposed here surpasses most similar systems in solving many more traditional benchmark problems and real-life problems. This, the system achieves by the combined impact of several small but important features such as powerful chromosome representation, effective genetic operators, restricted neighbourhood strategies and efficient search strategies along with innovative initial solutions. These features combined with the hybrid strategy employed enabled the system to solve several benchmark problems optimally, which has been discussed elsewhere in Meeran and Morshed (8th Asia Pacific industrial engineering and management science conference, Kaohsiung, Taiwan, 2007). In this paper we bring out the system's practical usage aspect and demonstrate that the system is equally capable of solving real life Job Shop problems.

S. Meeran (✉)
School of Management, University of Bath, Bath BA2 7AY, UK
e-mail: s.meeran@bath.ac.uk

M. S. Morshed
Birmingham, UK
e-mail: m.s.morshed@bath.ac.uk

Keywords Genetic algorithms · Tabu search · Job shop · Scheduling · Hybrid systems

Introduction and background to scheduling

In the present competitive world, scheduling plays a critical role in the competency of organisations. The solution to the job shop scheduling problem (JSSP) has a strong practical implication as many manufacturing organisations may fit the definition of a job shop. The JSSP problem is considered to be a good representation of the general domain of combinatorial problems and hence a proven solution strategy will have an impact on finding solutions to wide ranging problems. The generic nature of the problem, coupled with its reputation for being notoriously difficult to solve, (as it is considered to be an NP-hard problem), has led to the JSSP being considered as probably the most studied and well-developed problem in deterministic scheduling theory, serving as a comparative test-bed for different solution techniques.

The current wave of research in scheduling theory has evolved over the past fifty years and has resulted in a wide variety of papers discussing techniques that vary from unrefined dispatching rules to highly sophisticated bottleneck based heuristics. However with the advent of new methodologies, such as neural networks and evolutionary computation, researchers from fields such as computing, cognitive-biology and genetics have become important contributors, introducing a paradigm of intelligent solutions to this problem.

Job shop scheduling model

A job shop consists of a set of different machines that perform operations on production jobs. Each job has a number of operations which have a specified processing order through the machines and each operation has a specified processing time. Every job is independent. The job shop environment considered here does not allow pre-emption. The objective we set in this work is to find the best job sequence through the machines, which minimises the total completion time.

A number of researchers (Roy and Sussmann 1964; Adams et al. 1988; Blazewicz et al. 1996; Jones and Rabelo 1998; Jain and Meeran 1998b; Pinedo and Chao 1999) have discussed a variety of mathematical models of the problem. One of the factors which differentiate these models is the objective one wants to satisfy in producing the schedule. Obviously for different objectives, the schedules will be different. The most useful and common objective that the majority of researchers use is the make-span minimisation which has been also adapted for this research. A deterministic job-shop scheduling problem consists of a finite set J of n jobs $\{\mathcal{J}_i\}_{i=1}^{n}$ to be processed on a finite set M of m machines $\{\mathcal{M}_k\}_{k=1}^{m}$. Each job J_i must be processed on m_i number of machines and consists of a chain or complex of m_i operations $O_{i1}, O_{i2}, \ldots, O_{im_i}$ which have to be scheduled in a predetermined given order, a requirement called a precedence constraint. There are N operations in total, $N = \sum_{i=1}^{n} m_i \cdot O_{ik}$ is the operation of job J_i which has to be processed on machine M_k for an uninterrupted processing time period τ_{ik}. Furthermore, no operation may be pre-empted, i.e., interrupted and then completed at a later time. Each job has its own individual flow pattern through the machines, which is independent of the other jobs. The problem is further confined by capacity constraints (also called disjunctive constraints), which stipulate that each machine can process only one operation at a time. If the completion time of J_i on M_k is C_{ik} then the duration in which all the operations of all the jobs are completed is referred to as the make-span $\mathrm{C}_{\max}$. In the optimisation variant[1] of JSSP, the objective of the scheduler is to determine starting times for each operation, $t_{ik} \geq 0$, in order to minimise the make-span while satisfying all the precedence and capacity constraints. That is, the goal may be expressed (Jain and Meeran 2002) as determining $\mathrm{C}^*_{\max}$, where

$$\mathrm{C}^*_{\max} = min\,(\mathrm{C}_{\max}) = \underset{feasibleschedules}{min}\,(\max\,(t_{ik} + \tau_{ik}) : \forall\ \mathcal{J}_i \in \mathcal{J}, \mathcal{M}_k \in \mathcal{M})\,.$$

In spite of an unambiguous definition of operations given above, for simplicity we will be using a single subscripted notation in depicting operations in a schedule. However, the loss of information in removing one subscript is compensated by using implicit information that could be taken from the input data and from the order of sequence of the operations. Details of how it is carried out are given later in the paper.

[1] The optimisation variant is used to denote the problem itself and is distinguished from the decision variant of the problem which considers the question: Does there exist a solution in the search space that does not exceed a given upper bound?

Literature review

If one were to attempt an exact procedure to solve JSSP using exhaustive techniques (sometimes known as optimisation methods) such as linear programming, integer programming, mixed integer programming etc, the time to find a solution increases exponentially (or as a high degree polynomial) for a linear increase in problem size except for selected restricted versions (special cases) of the problem. (In fact, the general JSSP problem has been identified as an NP-hard problem). Although many improvements in the category of exact methods have been achieved through enhancing methods such as Branch and Bound (Mascis and Pacciarelli 2002; Tan et al. 2010), in general exhaustive techniques cannot be applied to large problems and their execution necessitates the need for a very good understanding of the job shop domain. Highly specialised inference rules and selection procedures are required to fathom nodes at high levels in the solution tree without explicit searching. Consequently, many researchers have turned their attention to approximation methods. However, the approximation methods do not guarantee achieving optimal solutions; rather they are able to attain near-optimal solutions. It is found that heuristics inspired by natural phenomena and intelligent problem solving are useful methods for JSSP solutions. The approximation techniques can be categorised into four major groups: priority dispatch rules (Panwalkar and Iskander 1977; Chiang and Fu 2007; Tay and Ho 2008), bottleneck based heuristics (Adams et al. 1988; Demirkol et al. 1997; Chen and Chen 2009; Zhang and Wu 2008), artificial intelligence and local search methods (which are discussed in more detail in the following paragraphs), all of which have many sub techniques within their groups. Many attempts have been made to apply these sub techniques to JSSP with varying success. Also further improvements on these techniques such as beam search and simulated annealing have enhanced the quality of the solutions that are based on the myopic selections such as the ones normally made by priority dispatch rules. However, the deviations from optimum are still high. Most of these existing approximation methods also have the weakness of needing high computing effort to achieve their best results. Some researchers (Fattahi et al. 2007) attempted combining approximation and exact techniques effectively in order to attain a better performance. However these techniques are also costly computationally.

The sub techniques of the approximation methods have increased in number and sophistication over the years. For example, the bottleneck based heuristic category lead to the shifting bottleneck procedure (SBP) (Adams et al. 1988) that has been found to be very effective in achieving substantial success in finding the best solutions for many problems. The primary weakness of this algorithm, however, is the high computing effort required. Another general weakness of the SBP approaches is the level of programmer sophistication needed.

In the case of the artificial intelligence (AI) category, sub techniques such as neural networks (Jain and Meeran 1998a; Weckman et al. 2008) expert systems and constraint satisfaction (Tan et al. 2010) have been applied to JSSP. However, the effectiveness of these techniques in finding optimum solutions has been limited.

Within the local search category many methods have been proposed by various researchers. They include algorithms such as simulated annealing (SA) (Satake et al. 1999; Suresh and Mohanasundaram 2005), genetic algorithms (GA) (Davis 1985; Della Croce et al. 1995; Dorndorf and Pesch 1995; Mattfeld 1996; Yamada and Nakano 1996; Park et al. 2003; Aydin and Fogarty 2004; Goncalves et al. 2005; Gao et al. 2006; Pezzella et al. 2008; Gholami and Zandieh 2009; Gen et al. 2009; Pérez et al. 2010), tabu search (TS) (Glover 1989; Dell'Amico and Trubian 1993; Nowicki and Smutnicki 1996; Thomsen 1997; Pezzella and Merelli 2000; Chen et al. 2007; Eswarmurthy and Tmilarasi 2009), ant optimisation and genetic local search (GLS) (Yamada and Nakano 1996; Zhou et al. 2009), scatter search and path relinking (SS&PR) (Jain and Meeran 2002). The majority of GA and GLS approaches appear to give poor results due to the difficulties they have with crossover operators and schedule encoding. Many of the computational studies indicate that in general TS provides the best results of all the local search techniques: most TS approaches are able to generate good schedules within reasonable computing time. However tabu search, like many other local search methods, requires many parameters to be carefully fine-tuned and appropriately adjusted for each problem. These requirements are difficult to achieve. In brief, although considerable progress has been made in recent years the inherent difficulty faced by all of these methods is that no heuristic with a guaranteed performance has been developed so far. For most approximation methods there are instances in which they perform badly as it is not completely known under which given circumstances a procedure is likely to succeed or fail. It is therefore apparent that if the current barriers within job shop scheduling problems are to be overcome, hybrid approaches are worth considering.

A popular solution structure of choice hence has been a hybrid construction amalgamating several methods with a meta-strategy to guide a myopic local heuristic to the global optimum [Yu and Liang 2001 (NN&GA); Wang and Zheng 2001 (GA&SA); Park et al. 2003 (parallel GA (PGA)); Goncalves et al. 2005 (GA&PDR); Rossi and Boschi 2009 (Ant GA); Zhang and Gen 2009 (Hybrid Swarm optimisation)]. Although in general hybridisation improves performance, these hybrids do require substantial computing time. Another problem with hybrids is that there is no formal method to suggest effective ways of combining heuristic techniques; hence there is a need for exploring various combinations of search techniques. The work reported here is one such attempt using clever ways of combining the search techniques GA and TS along with intelligent subtechniques. There have been a small number of systems that use the combination of GA and TS in providing a solution to JSSP (Meeran and Morshed 2007; Tamilselvan and Balasubramanie 2009; González et al. 2009; Chiu et al. 2007; Zhang et al. 2010). Tamilselvan and Balasubramanie (2009)) have used GA as the base search mechanism and TS to improve their search. They have demonstrated the effectiveness of the combination of GA and TS, which is called GTA against standalone GA and TS using a limited number of example problems that they have devised. There is no evidence presented of the system being tested on established benchmark problems or on real life practical problems. González et al. (2009) presented a hybrid GA and TS system as in the case of Meeran and Morshed (2007), however Gonzalez et al's proposed method is for the job shop scheduling problem with set-up times. Although they have obtained some very good results, their proposed system is for different set of bench mark problems and also they have reported results of a limited number of established benchmark problems, namely 6 instances of LA and three instances of ABZ. Most other systems (Chiu et al. 2007; Zhang and Wu 2008) shown a good progress in solving a specific set of benchmark problems albeit in some cases the benchmark problems used are not from the hard instances of established benchmark problems such as LA, ABZ, ORB and FT. Furthermore, it could not be established from the publications that most of these systems work well with real life practical problems in addition to solving standard JSSP benchmark problems. The system being presented here is tested on a substantial number of bench mark problems including hard instances from FT, LA, ABZ and ORB, attaining optimum solutions for 48 out of 51 of them. Details of the results attained are available in Meeran and Morshed (2007). As mentioned earlier, here we are presenting in this paper another aspect of the system with regard to its application to real life practical cases from real life manufacturing companies.

System developed

This paper presents a framework using a hybrid technique involving GA and TS to find a non-problem-specific solution to the JSSP. The system's objective is to minimise the make-span (C_{max}) criterion while satisfying all constraints.

GA has the ability to do a parallel search to explore the global search space. Through the parallel search mechanism

GA retains useful information about what has been learned from previous generations. GA differs from traditional optimisation in several ways such as working with the coding of decision variables rather than the variables themselves. It searches from a population of points rather than a single point and its algorithm is computationally simple but powerful. On the other hand TS works on individual strings, which are points on the solution space. TS guides (Glover 1989; Barnes and Chambers 1995) the iterations from one neighbourhood point to another by locally improving the solution's quality and has the ability to avoid poor local minima. A hybrid system combining both GA and TS using their complementary strengths has a good chance of providing a reasonable solution to global combinatorial optimisation problems such as JSSP.

In the model presented here (Fig. 1), during the hybrid search process, GA starts with a set of initial solutions and generates a set of new solutions, on each of which TS performs a local search to improve them. Then GA uses the TS improved solutions to continue with parallel evolution. A restricted list is used in TS to prevent re-visiting the solution previously explored. This hybrid framework can be converted into traditional GA by omitting the TS steps. Similarly, it can be converted into traditional TS by setting the population size to one and omitting the genetic operators. Such a hybrid strategy preserves the generality of GA and TS and can be easily adapted to other combinatorial and functional optimisation problems. The details and the results obtained for standalone GA and TS systems are given in Morshed (2006).

The initial set of solutions necessary for GA can be generated by different methods such as mid-cut and priority dispatch rules (PDR). The mid-cut heuristic for the initial solution string works as follows. From the input constraint and precedence relations, a sequential chain of operations is created. This string is cut at a position between the first and the last positions creating two sub-strings of operations. The first sub-string is moved to the back of the second sub-string, which effectively maintains the total number of operations in a string and creates a new solution string. This process will continue until each job is at the first position of a solution string. Thus the total number of solution strings created using this method equals the total number of jobs. These strings as initial solutions have been found to be better than the initial solutions created by the random method or priority despatch rules.

GA requires an appropriate chromosome (i.e. a string of operations) to represent a solution (Cheng et al. 1996). It is then essential to ensure that all chromosomes, which are generated during the evolutionary process, represent feasible solutions. In the classic JSSP, if one considers there are J jobs to be processed on M machines resulting in a total number of $J \times M$ operations then a chromosome $[gene_1, gene_2, gene_3 \ldots, gene_{J\times M}]$ can represent such a schedule, which could be an arbitrary list of $J \times M$ operations (in operation based representation). The schedule could be constructed according to the sequence of operations that appears in the string using certain heuristics. Once a schedule could be represented by such a chromosome there is a need for genetic operators such as crossover and mutation. A general crossover operator of GA operates on two parents' strings at a time and generates offspring strings by recombining both parent strings' features. This operator needs to preserve job sequence characteristics of the parent's string in the next generation. If good features of different strings are properly combined, the offspring strings generated may have even better features. How the chromosome that is used in this work preserves the operation precedence is explained below.

There are two kinds of order relations in the job shop scheduling problem: firstly, the operation sequence on the machine; secondly, the precedence constraint among operations of a job. The first kind will be determined by a solution method, while the second one is the requirement to be maintained in the schedule. Hence the information with respect to both operation sequence and precedence constraints has to be preserved in an encoding simultaneously. In such a case, many crossover methods dealing with literal permutation encodings cannot deliver that. A chromosome may be either infeasible, in the sense that some precedence constraints are violated, or illegal and the number of repetitions of some symbols (i.e. operation or job numbers) in the resulting chromosomes may not equal the prescribed numbers.

The chromosome representation for the JSSP used here based on Gen et al. (1994) and Cheng et al. (1996) encodes a schedule (solution) as an ordered sequence of job/operations, where each gene stands for one operation. In this representation, operations are listed in the order in which they are scheduled. For an n—job and m—machine problem, a chromosome contains '$n \times m$' genes. Each job appears in the chromosome exactly m times and each repeating gene does not indicate a specific operation of a job but refers to an operation of a job which is context dependent that is defined by the position the gene occupies in the string. Let us use an example of 4×4 Job shop problem to explain this. The Scheduling problem is normally given as a set of precedence constraints along with the operation processing times as shown in Tables 1 and 2. For example, J1 has to be processed by M1, M2, M3 and M4 respectively in that sequence. The task of a solution method is to find the sequence in which these operations had to be scheduled on the machines taking into account the precedence constraints and the processing times of the operations with the aim to optimise an objective function such as completing all operations in the quickest time possible. In other words the aim is to find the job sequence

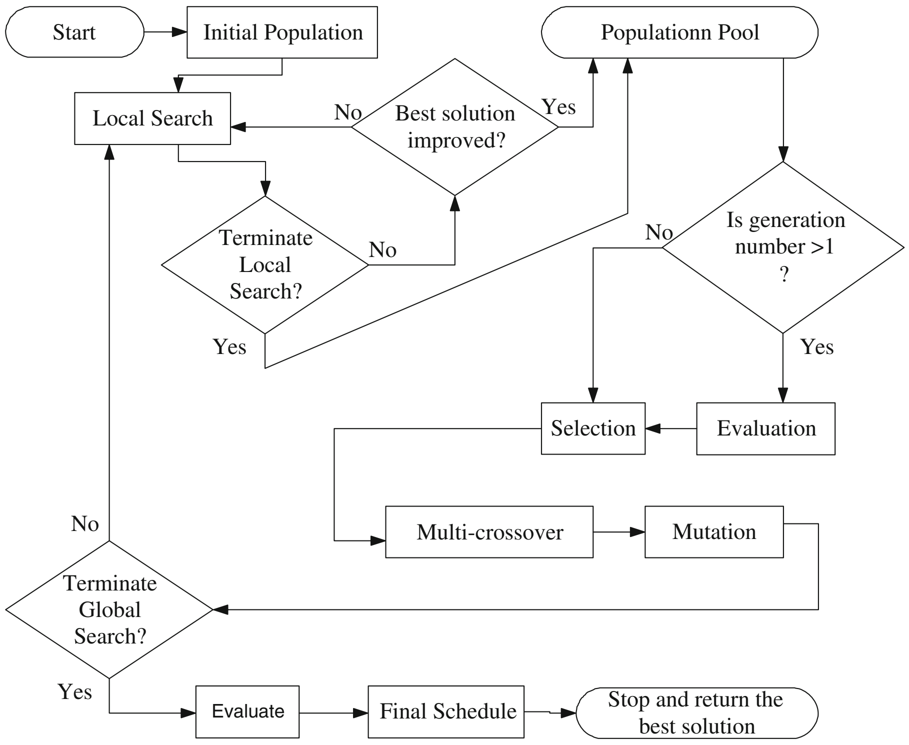


Fig. 1 Model of the system proposed

Table 1 The machine sequence matrix

	Operations			
Job	1	2	3	4
J1	M1	M2	M3	M4
J2	M3	M1	M4	M2
J3	M1	M3	M2	M4
J4	M3	M2	M4	M1

Table 2 Example of a four-job four-machine job shop problem

Processing time				
Job	Operations			
	1	2	3	4
J1	3	3	2	2
J2	1	5	3	2
J3	3	2	3	4
J4	2	4	3	3

Ji represents the jobs, Mi represents the machines

Table 3 Job sequence—for the example four-job four-machine job shop problem

Machine	Operations			
	1	2	3	4
M1	J3	J1	J2	J4
M2	J4	J3	J1	J2
M3	J4	J3	J2	J1
M4	J4	J3	J2	J1

for every machine, which is given traditionally in the form of Job sequence matrix as shown in the Table 3.

The main concern here is to find the correspondence between the traditional representation of schedules and the chromosomes that we would like to represent the schedule with, so that we can use the GA to evolve these chromosomes to find better schedules. From the above schedule a chromosome such as [4 3 3 4 3 4 3 1 2 2 2 1 4 1 1 2] could be derived which in turn could be changed into a possible schedule of operations. Here in this chromosome the genes '1' stand for J1 showing Job1 has four operations, the order of these operations correspond to the relative positions of the genes '1' i.e. their order of occurrence in the chromosome. Likewise other genes could be interpreted. The first gene '4' corresponds to the first operation of J4 and the second gene '3' corresponds to the first operation of J3 and so on. Hence any permutation of the chromosome always yields a feasible schedule. The information on machine requirements for different operations could be extracted from the input data of the problem such as the precedence constraints. e.g. the last gene 2 which represents the fourth operation of J2, the machine needed for this operation can be seen as Machine 2. The machine requirements for all the operations can be gathered in a list which are linked to the operations i.e. genes in the chromosome. As mentioned earlier as the genes are represented generically and not anchored to a particular order or position, the relative order between them is always valid whatever actual positions the genes occupy

Chromosome: [4 3 3 4 3 4 3 1 2 2 2 1 4 1 1 2]

Shadow Machine

Chromosome: $[4_3\ 3_1\ 3_3\ 4_2\ 3_2\ 4_4\ 3_4\ 1_1\ 2_3\ 2_1\ 2_4\ 1_2\ 4_1\ 1_3\ 1_4\ 2_2]$

Fig. 2 Processing order of jobs on machines (machine list for chromosome)

in the chromosome. The precedence constraints are always preserved whatever permutations a particular gene takes. In this example such a list is given as [3 1 3 2 2 4 4 1 3 1 4 2 1 3 4 2]. The chromosome representing the schedule of operations and the list of the machines are linked by a mechanism of a shadow chromosome which operates in the background. Such a shadow Machine chromosome for this example is $[3_2 1_3 3_2 2_4 2_3 4_3 4_4 1_3 3_1 1_5 4_3 2_3 1_3 3_2 4_2 2_2]$.

The next part of our discussion is how we will formulate a schedule (and represent the same in a format such as Gantt chart) given a chromosome. Let us consider the chromosome [4 3 3 4 3 4 3 1 2 2 2 1 4 1 1 2] we have been looking at. The first operation (gene) of the chromosome is scheduled first and then the second operation in the chromosome is considered and the process continues. Each operation under consideration is allocated to the earliest available position on the machine which processes this operation. The process is repeated until all operations in the chromosome are scheduled. It can be seen that the actual schedule generated in this manner is guaranteed to be an active schedule. Now from the machine list or from the shadow machine chromosome we can deduce the sequence operations for each machine. This is done by simply scanning the machine list for a particular machine say Machine 1 (refer to Fig. 2) and finding the corresponding operations in the schedule chromosome. In this example it can be seen that the operation sequence (job processing order) for Machine 1 is [J3, J1, J2, J4]. Like wise the job sequence could be extracted for other machines. Further a full schedule can be made by supplementing the job sequences with processing times taken from the input data.

Having established how the chromosomes and schedules are linked let us look at the genetic operators. The construction of the relative order crossover operator for JSSP can be described as follows: at first, two parents are chosen and then 'q' number of jobs in these parents are selected randomly (Note the 'q' is a random integer number which is greater than one and less than the total number of jobs for any JSSP instance). These q selected jobs correspond to the $q \times m$ operations in each parent. A string is produced from the first parent by picking out these operations with their order unchanged while filling other positions with holes (E). Then the string is rotated forward or backward by r positions (where r is a random integer number) maintaining the relative positions of individual genes. Then the holes (E) are filled with the not-selected operations of the second parent, one by one, while the relative order of the not-selected operation is maintained simultaneously. One child is produced this way. Similarly, the other child can be produced with the selected operations of the second parent and non-selected operations of the first.

The chromosome (solution string) represented below is indexed with job number and its corresponding machine. For the first parent $parent_1$, before undergoing any genetic operation the scheduling had a $C_{\max}$ value of 28 (Fig. 3).

$parent_1 = [2, 2, 3, 4, 3, 1, 4, 3, 1, 1, 2, 2, 1, 4, 3, 4]$ and corresponding machine index (shadow chromosome) of these operations is as follows:

$$[2_3\ \ 2_1\ \ 3_1\ \ 4_3\ \ 3_3\ \ 1_1\ \ 4_2\ \ 3_2\ \ 1_2\ \ 1_3\ \ 2_4\ \ 2_2\ \ 1_4\ \ 4_4\ \ 3_4\ \ 4_1]$$

The second parent, $parent_2$, before undergoing any genetic operation also had a $C_{\max}$ of 28.

$parent_2 = [1, 1, 4, 3, 1, 3, 2, 4, 4, 2, 1, 2, 2, 3, 4, 3]$ and corresponding machine index of these operations is as follows:

$$[1_1\ \ 1_2\ \ 4_3\ \ 3_1\ \ 1_3\ \ 3_3\ \ 2_3\ \ 4_2\ \ 4_4\ \ 2_1\ \ 1_4\ \ 2_4\ \ 2_2\ \ 3_2\ \ 4_1\ \ 3_4]$$

Step 2 of the algorithm selects jobs randomly, for example, in this case J1 and J3 are selected in $parent_1$: (which are shown below boldfaced in the chromosome)

$parent_1 = [2, 2, \mathbf{3}, 4, \mathbf{3}, \mathbf{1}, 4, \mathbf{3}, \mathbf{1}, \mathbf{1}, 2, 2, \mathbf{1}, 4, \mathbf{3}, 4]$.

In step 3 the not-selected operations are filled with 'E' and hence from $parent_1$ the following string $string_1$ is produced.

$string_1 = [E, E, 3, E, 3, 1, E, 3, 1, 1, E, E, 1, E, 3, E]$

Similarly, $string_2$ shown below is produced from $parent_2$.

$parent_2 = [\mathbf{1}, \mathbf{1}, 4, \mathbf{3}, \mathbf{1}, \mathbf{3}, 2, 4, 4, 2, \mathbf{1}, 2, 2, \mathbf{3}, 4, \mathbf{3}]$

$string_2 = [\mathbf{1}, \mathbf{1}, E, \mathbf{3}, \mathbf{1}, \mathbf{3}, E, E, E, E, \mathbf{1}, E, E, \mathbf{3}, E, \mathbf{3}]$

In step 4, two sub-strings are produced from both parents respectively having only the not-selected operations of their respective parents ($SubString_1$[2, 2, 4, 4, 2, 2, 4, 4] from $parent_1$ and $SubString_2$ [4,2,4,4,2,2,2,4] from $parent_2$).

In step 5, the $string_1$ is rotated one position forward and $string_2$ is rotated 'zero' positions (i.e. left as such) to make $string_3$ and $string_4$ respectively:

$string_3 = [E, \mathbf{3}, E, \mathbf{3}, \mathbf{1}, E, \mathbf{3}, \mathbf{1}, \mathbf{1}, E, E, \mathbf{1}, E, \mathbf{3}, E, E]$ and its corresponding machine index is as follows:

$$string_3 = [E\ \ 3_1\ \ E\ \ 3_3\ \ 1_1\ \ E\ \ 3_2\ \ 1_2\ \ 1_3\ \ E\ \ E\ \ 1_4\ \ E\ \ 3_4\ \ E\ \ E]$$

$string_4$=[**1**, **1**, E, **3**, **1**, **3**, E, E, E, E, **1**, E, E, **3**, E, **3**] and its corresponding machine index as follows:

$$string_4 = [1_1\ \ 1_2\ \ E\ \ 3_1\ \ 1_3\ \ 3_3\ \ E\ \ E\ \ E\ \ E\ \ 1_4\ \ E\ \ E\ \ 3_2\ \ E\ \ 3_4]$$

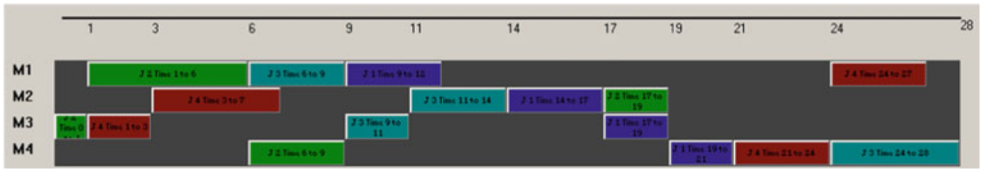


Fig. 3 Machine Gantt chart for parent1 (C_{max} is 28)

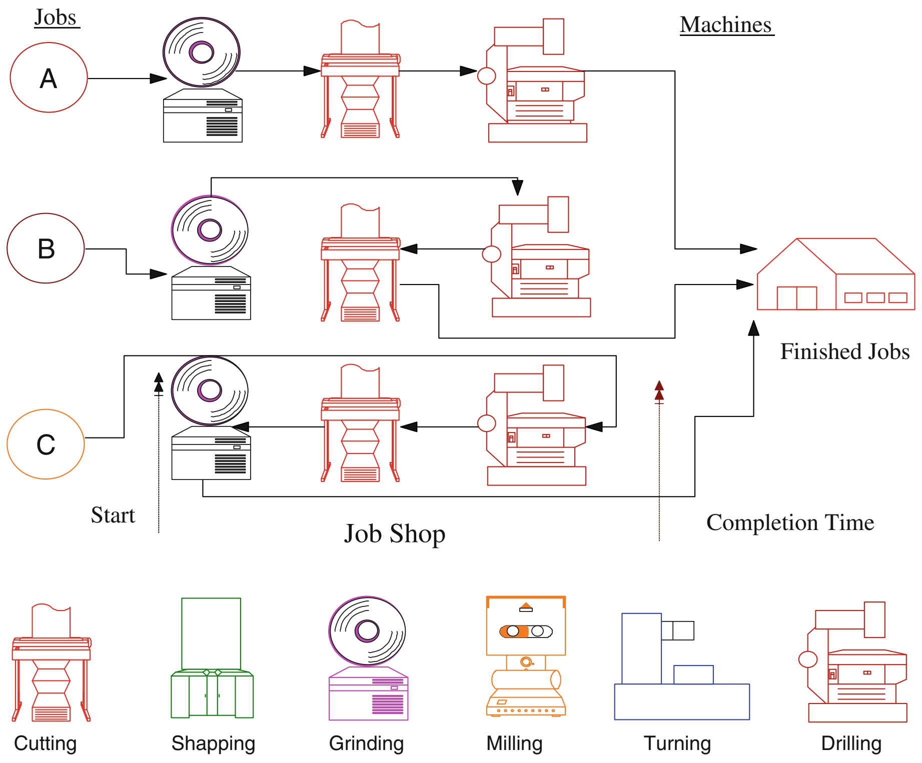


Fig. 4 Machines and typical machine layout of work shops at the steel mill

In step 6, the 'child' strings are produced using the relative order crossover mechanism where the relative position of the operations for any job is maintained. This is indicated by the machine index (i.e. the relative positions of machine indices remain the same, even though the operations have moved due to applying genetic crossover), of these created strings. Filling the holes of $string_3$ with the not-selected operations of $SubString_2$[4, 2, 4, 4, 2, 2, 2, 4] from $parent_2$, has produced one offspring: $child_1$ and the corresponding C_{max} is 23, which is an improvement due to the crossover operation.

$child_1$ = [**4**, 3, **2**, 3, 1, **4**, 3, 1, 1, **4**, **2**, 1, **2**,3, **2**, **4**] and its corresponding machine index is as follows:

$$child_1 = [4_3 \quad 3_1 \quad 2_3 \quad 3_3 \quad 1_1 \quad 4_2 \quad 3_2 \quad 1_2 \quad 1_3 \quad 4_4 \quad 2_1 \quad 1_4 \quad 2_4 \quad 3_4 \quad 2_2 \quad 4_1]$$

Similarly, filling the holes of $string_4$ with the not-selected operations of *SubString1* [2, 2, 4, 4, 2, 2, 4, 4] from $parent_1$, has produced a second offspring: $child_2$ and a corresponding C_{max} of 25.

$child_2$ = [1, 1, **2**, 3, 1, 3, **2**, **4**, **4**, **2**,1, **2**, **4**, 3, **4**, 3] and its corresponding machine index is as follows:

$$child_2 = [1_1 \quad 1_2 \quad 2_3 \quad 3_1 \quad 1_3 \quad 3_3 \quad 2_1 \quad 4_3 \quad 4_2 \quad 2_2 \quad 1_4 \quad 2_4 \quad 4_4 \quad 3_2 \quad 4_1 \quad 3_4]$$

The second genetic operator, mutation, can help GA to get a better solution in a faster time. In this model, relocation is used as a key mechanism for mutation. Operations of a particular job that is chosen randomly are shifted to the left or to the right of the string. Hence the mutation can introduce diversity without disturbing the sequence of a job's operations. When applying mutation one has to be aware that if the diversity of the population is not sufficiently maintained, early convergence could occur and the crossover cannot work well.

As mentioned earlier, the local improvement procedure used in this framework is based on TS. In essence TS is a simple deterministic search procedure that transcends local optimality by storing a search history in its memory. In TS, a function (called a move) that transforms a solution string into another solution string is defined. Hence for any solution string, S, a subset of moves applicable will form the neighbourhood of S. Four different types of genetic neighbourhood generation techniques have been identified,

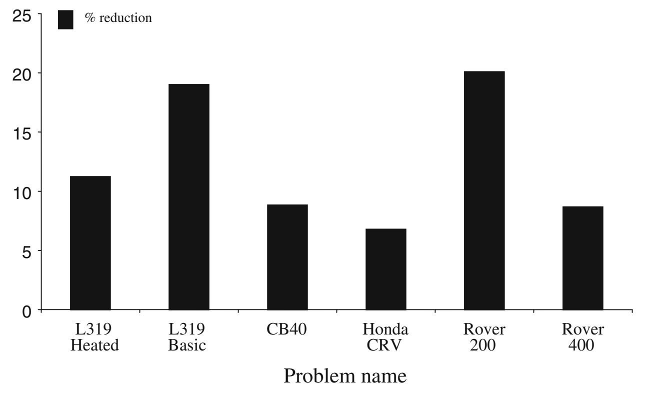


Fig. 5 Improvement in schedule lengths at the automobile windscreen producer

Table 4 Test cases from steel mill

Problem name	Descriptions	Size
JSSP1	8 Jobs and 6 machines job shop problem	8 × 6
JSSP2	6 Jobs and 6 machines job shop problem	6 × 6
JSSP3	6 Jobs and 6 machines job shop problem	6 × 6

developed and used in the model along with existing critical path based neighbourhood structures such as the one proposed by Nowicki and Smutnicki (1996). The model directly works on these neighbourhood structures with minimum manipulation of the solution strings. The following explains how the system deals with the second type. More details on this and other local search techniques used are given in Morshed (2006).

A solution string, containing all operations to be scheduled, needs to identify its critical operations by using simple earliest and latest operation completion times. Once the critical path of operations is identified for any string, the system needs to find all critical blocks (i.e. critical operations bunched together without any idle time on the same machine). The move function is applied on operations within the critical blocks. Then, as in the case of generic TS philosophy, the new solution string after a move will create a neighbour of the original solution string, which is a new point on the search space. Every time the move function gets a new improved solution string this will be kept in the memory for further exploration until the total number of iterations is met or an optimal solution string is found. It is worth noting that TS is directly applied, in all cases of neighbourhoods in this system, on the solution strings of the solution pool resulted from GA to improve their quality.

Results

The system is used to find the make-spans of the benchmark problems, many of which are 'square problems'. The system has done well in finding solutions to these problems as it is well known that most of the square problems are harder to solve than rectangular problems. The system found optimum solutions for 48 of these benchmark problems and achieved an average mean relative error (MRE) of 0.08%. 25 runs with a population size of 200 are conducted to achieve this result in 300 generations on the GA side of the system. On the TS side a tabu length varying between 6 and 36 and a maximum number of iterations of 100 are used. If the system is allowed to run for more generations it is highly likely that it will find better solutions for the difficult problems. The detailed results which are overwhelmingly better for the system proposed here than many other comparable systems are presented elsewhere (Meeran and Morshed 2007). Here we are presenting the system's performance on real life practical problems.

For the practical implementation of the system, two case studies have been conducted in two companies which belong to the steel mills sector and the automobile windscreen processing industry respectively. In the case of the steel mill the system is tested on general job shops and in the case of the automobile windscreen producer the system is tested on general flow shops.

The steel mill produces spare parts required for its rolling mills in its medium sized workshops and is busy all year round. Each of these job shops has a number of machines, including cutting machines, shaper machines, grinding machines, milling machines, lathes and special turning machines, drilling/boring machines, polishing machines, painting and drying shops. These shops produce heavy-duty parts which are processed in different machines. Common raw materials are used for the production of these spares in special manufacturing processes and work is also carried out on bought out semi-finished products for particular jobs. Currently the company uses a simple manual scheduling system. A typical workshop in the company is shown in Fig. 4.

The automobile windscreen producer is one of the world's largest manufacturers of glass and glazing products for building, automotive and related technical markets. The factory in the United Kingdom produces 6,000 pieces of automobile windscreen per week in 92 variations (original equipment derivatives) through two different tracks called 'A' and 'B', each of them having three workstations. The factory also has orders for reengineering work on varieties of products as well as for replacement of broken items. Currently the company does its scheduling by using simple spread-sheets.

In both the above cases (the steel mill as well as the automobile windscreen producer) the system presented here found a far superior scheduling solution in quality as well as in computing time in comparison to the current systems operated by the companies. In achieving this performance the system used the same parameters as were used in the case of benchmark problems. Figure 5 shows the percentage reduction achieved by the system in 'schedule lengths' for

Table 5 JSSP1 test case and results

J	O_1		O_2		O_3		O_4		O_5		O_6	
	M	P	M	P	M	P	M	P	M	P	M	P
1	1	10	2	32	3	21	4	40	5	53	6	23
2	2	35	1	0	3	29	4	51	5	48	6	20
3	4	54	1	0	5	53	3	0	6	21	2	0
4	3	38	2	23	4	65	5	61	6	21	1	0
5	1	15	2	30	3	31	4	54	5	53	6	18
6	1	16	2	25	3	17	4	68	5	64	6	0
7	4	54	1	0	5	53	3	0	6	21	2	0
8	2	28	1	16	3	22	4	55	5	59	6	20

Generation	Makespan (C_{max})	MRE %
0	539 (lower) and 857 (upper)	6.31 and 41.1
30	505	0.00
60	505	0.00
90	505	0.00

J = job, M = machine, P = processing time, O = operation

various real life cases of the automobile windscreen producer when compared to what is achieved by the existing system.

Again, the presentation of the performance of the system in this paper is more focused on the former case of general job shops in the steel mill. Hence a deeper insight into what is involved in these job shops is provided below. Four types of products are currently being manufactured at this facility. They are (1) mild steel deformed bars, angles and steel channels, (2) pipes of different diameter, (3) low carbon wire rods, torsteel bar, plain round wrought iron rods and bars and (4) casting spare parts for the automobile and agricultural industries.

The company uses steel scrap that is mixed with scrap obtained from ship-breaking yards. This selected proper charge mix is then melted in high-frequency induction melting furnaces to obtain a homogenous molten metal. Appropriate quantities of ferro-alloys are then added to the molten metal. Through a series of other technical processes, and after careful chemical control, the product of these induction-melting furnaces is rolled in a 16-inch roughing mill to obtain billets of the desired size and specifications. These billets are then fed into a 10-inch finishing mill that is equipped with repeaters in order to ensure a continuous production line of deformed bars.

The company also manufactures pipes, mainly used for modern water supply and irrigation systems. The pipe mill has a precision-engineered welding plant and a galvanising section containing a zinc kettle. The plant uses zinc slabs of near-perfect purity. The company endeavours to match products to buyer-driven specifications in terms of steel grade, tensile strength, unit mass and dimensional tolerances.

Implementation of the proposed system in the steel mill's test cases

The system proposed here is tested on three job shops in the steel mill. Table 4 shows the profiles of the job shops (and their names for further reference). In all these cases the system is able to attain improved solutions in a reasonable computational time. In the following subsections the analysis of performance of the system has been provided outlining the impact on the make-span of these different real life problems.

Test case 1-JSSP1

Table 5 provides the problem data and results for the test case JSSP1. Figure 6 shows the schedules obtained using the proposed system in the form of Gantt charts. Although the initial generation model to start with has the maximum MRE of 41.1% the optimal solution for this problem was achieved even before the 30th generation of search. The current scheduling length the company operates, which is obtained using the existing scheduling system, is more than four days and most times several machines remain idle during the overall job processing. The company could save time and increase productivity by simply implementing the proposed system to schedule their jobs. The minimum make-span achieved using the proposed system for this problem is 505 units of time.

Test case 2-JSSP2

Table 6 provides the problem data and results for the test case JSSP2. Figure. 7 shows the schedules obtained for this test case using the proposed system in the form of Gantt charts.

a

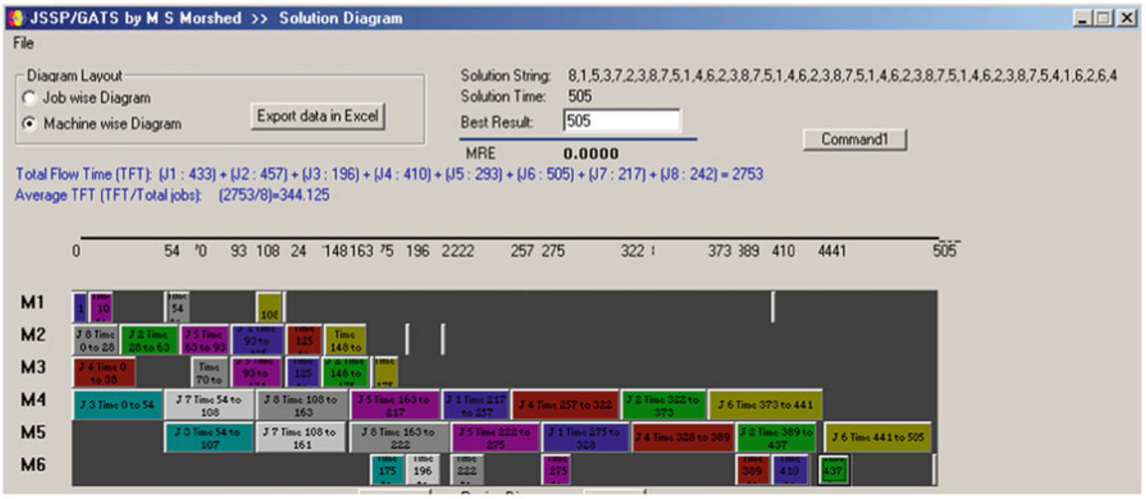


Diagram Layout: Job wise Diagram [✓ Machine wise Diagram]

Solution String: 8,1,5,3,7,2,3,8,7,5,1,4,6,2,3,8,7,5,1,4,6,2,3,8,7,5,1,4,6,2,3,8,7,5,1,4,6,2,3,8,7,5,4,1,6,2,6,4
Solution Time: 505
Best Result: 505
MRE 0.0000

Total Flow Time (TFT): (J1 : 433) + (J2 : 457) + (J3 : 196) + (J4 : 410) + (J5 : 293) + (J6 : 505) + (J7 : 217) + (J8 : 242) = 2753
Average TFT (TFT/Total jobs): (2753/8)=344.125

b

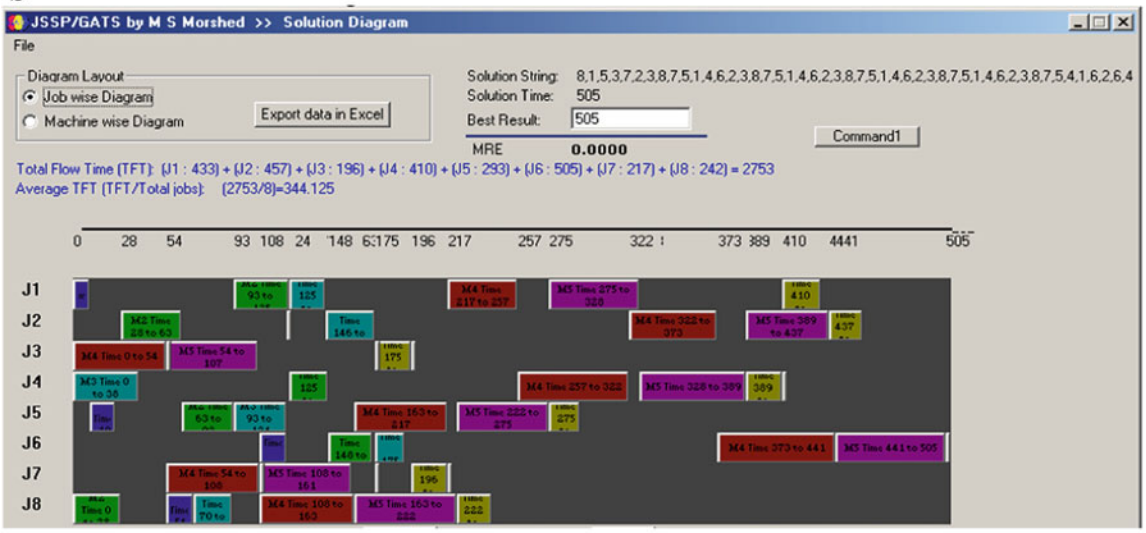


Diagram Layout: [✓ Job wise Diagram] Machine wise Diagram

Solution String: 8,1,5,3,7,2,3,8,7,5,1,4,6,2,3,8,7,5,1,4,6,2,3,8,7,5,1,4,6,2,3,8,7,5,1,4,6,2,3,8,7,5,4,1,6,2,6,4
Solution Time: 505
Best Result: 505
MRE 0.0000

Total Flow Time (TFT): (J1 : 433) + (J2 : 457) + (J3 : 196) + (J4 : 410) + (J5 : 293) + (J6 : 505) + (J7 : 217) + (J8 : 242) = 2753
Average TFT (TFT/Total jobs): (2753/8)=344.125

c

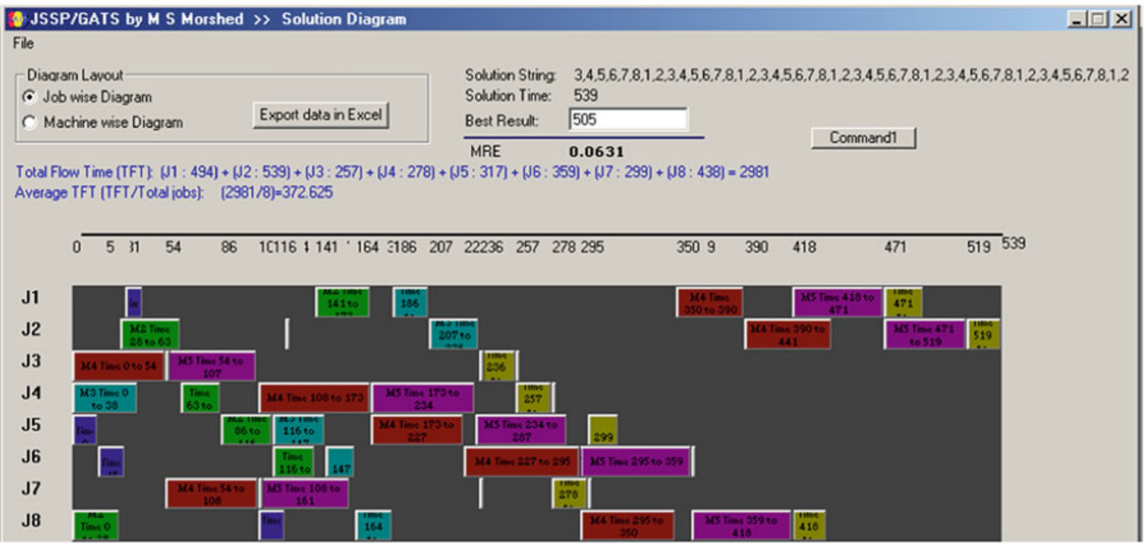


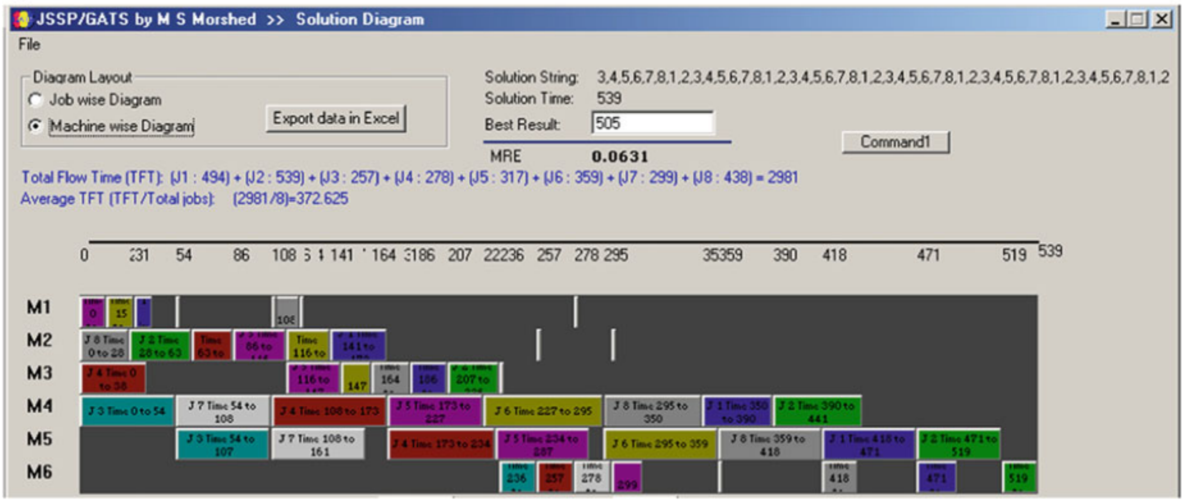
Diagram Layout: [✓ Job wise Diagram] Machine wise Diagram

Solution String: 3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2
Solution Time: 539
Best Result: 505
MRE 0.0631

Total Flow Time (TFT): (J1 : 494) + (J2 : 539) + (J3 : 257) + (J4 : 278) + (J5 : 317) + (J6 : 359) + (J7 : 299) + (J8 : 438) = 2981
Average TFT (TFT/Total jobs): (2981/8)=372.625

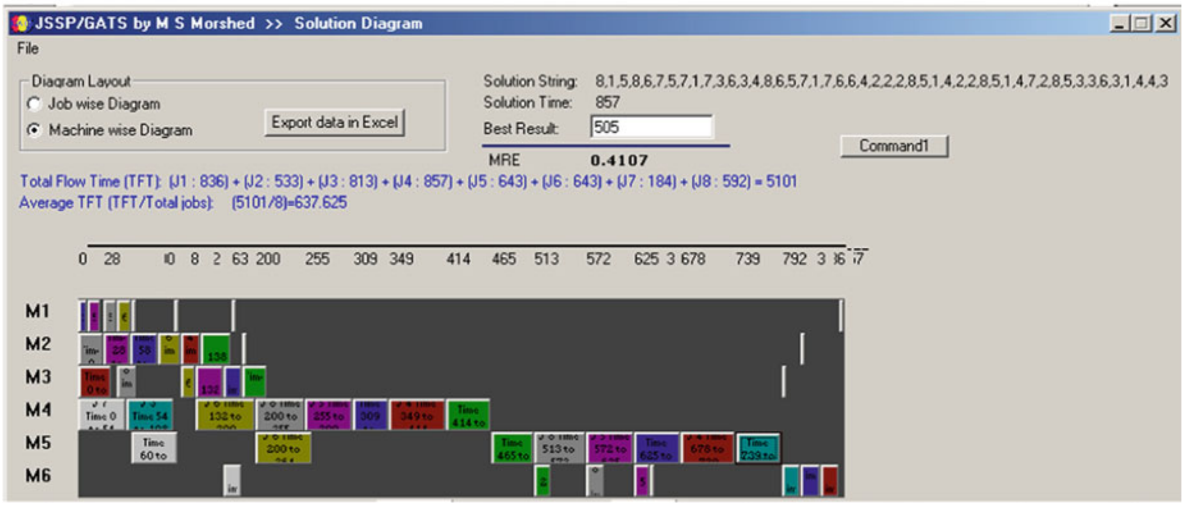
Fig. 6 **a** Machine Gantt chart for JSSP1 test case. **b** Job Gantt chart for JSSP1 test case. **c** Job Gantt chart for JSSP1 test case with C_{max} 539. **d** Machine Gantt chart for JSSP1 test case with C_{max} 539. **e** Machine Gantt chart for JSSP1 test case with C_{max} 857

d



Solution String: 3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2
Solution Time: 539
Best Result: 505
MRE 0.0631
Total Flow Time (TFT): (J1 : 494) + (J2 : 539) + (J3 : 257) + (J4 : 278) + (J5 : 317) + (J6 : 359) + (J7 : 299) + (J8 : 438) = 2981
Average TFT (TFT/Total jobs): (2981/8)=372.625

e



Solution String: 8,1,5,8,6,7,5,7,1,7,3,6,3,4,8,6,5,7,1,7,6,6,4,2,2,2,8,5,1,4,2,2,8,5,1,4,7,2,8,5,3,3,6,3,1,4,4,3
Solution Time: 857
Best Result: 505
MRE 0.4107
Total Flow Time (TFT): (J1 : 836) + (J2 : 533) + (J3 : 813) + (J4 : 857) + (J5 : 643) + (J6 : 643) + (J7 : 184) + (J8 : 592) = 5101
Average TFT (TFT/Total jobs): (5101/8)=637.625

Fig. 6 continued

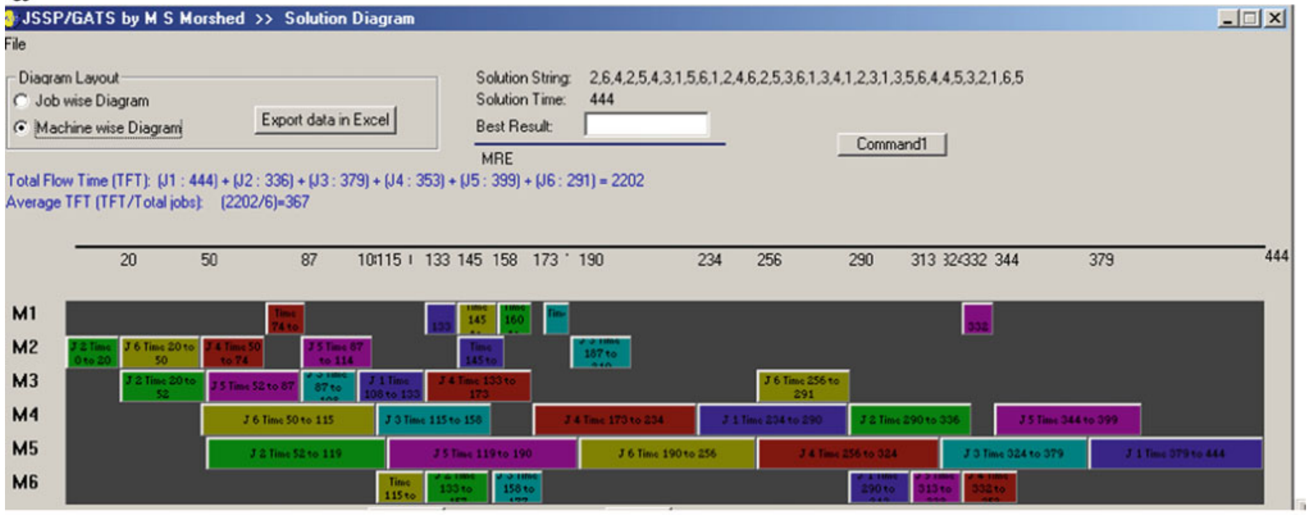
Table 6 JSSP2 test case and results

J	O_1		O_2		O_3		O_4		O_5		O_6	
	M	P	M	P	M	P	M	P	M	P	M	P
1	3	25	1	12	2	18	4	56	6	23	5	65
2	2	20	3	32	5	67	6	24	1	13	4	46
3	3	21	4	43	6	19	1	10	2	23	5	55
4	2	24	1	15	3	40	4	61	5	68	6	21
5	3	35	2	27	5	71	6	19	1	12	4	55
6	2	30	4	65	6	18	1	15	5	66	3	35

Generation	Makespan (C_{max})	MRE %
0	493 (lower) and 616 (upper)	9.94 and 27.92
30	444	0.00
60	444	0.00
90	444	0.00

J = job, M = machine, P = processing time, O = operation

a



JSSP/GATS by M S Morshed >> Solution Diagram

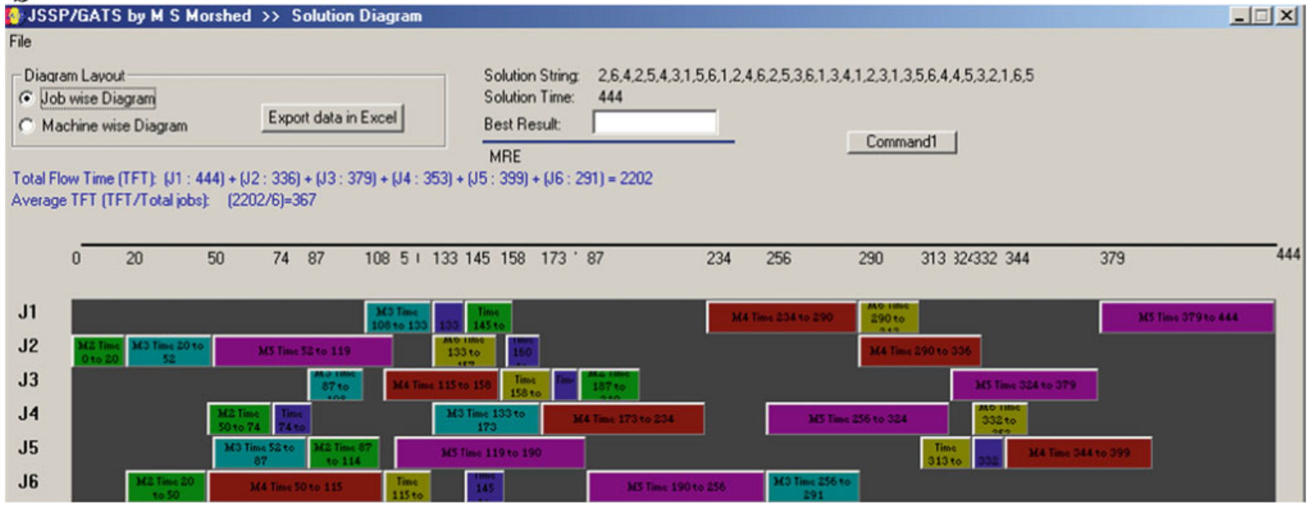
File

Diagram Layout: Job wise Diagram [✓ Machine wise Diagram]

Solution String: 2,6,4,2,5,4,3,1,5,6,1,2,4,6,2,5,3,6,1,3,4,1,2,3,1,3,5,6,4,4,5,3,2,1,6,5
Solution Time: 444
Best Result:
MRE

Total Flow Time (TFT): (J1 : 444) + (J2 : 336) + (J3 : 379) + (J4 : 353) + (J5 : 399) + (J6 : 291) = 2202
Average TFT (TFT/Total jobs): (2202/6)=367

b



JSSP/GATS by M S Morshed >> Solution Diagram

File

Diagram Layout: [✓ Job wise Diagram] Machine wise Diagram

Solution String: 2,6,4,2,5,4,3,1,5,6,1,2,4,6,2,5,3,6,1,3,4,1,2,3,1,3,5,6,4,4,5,3,2,1,6,5
Solution Time: 444
Best Result:
MRE

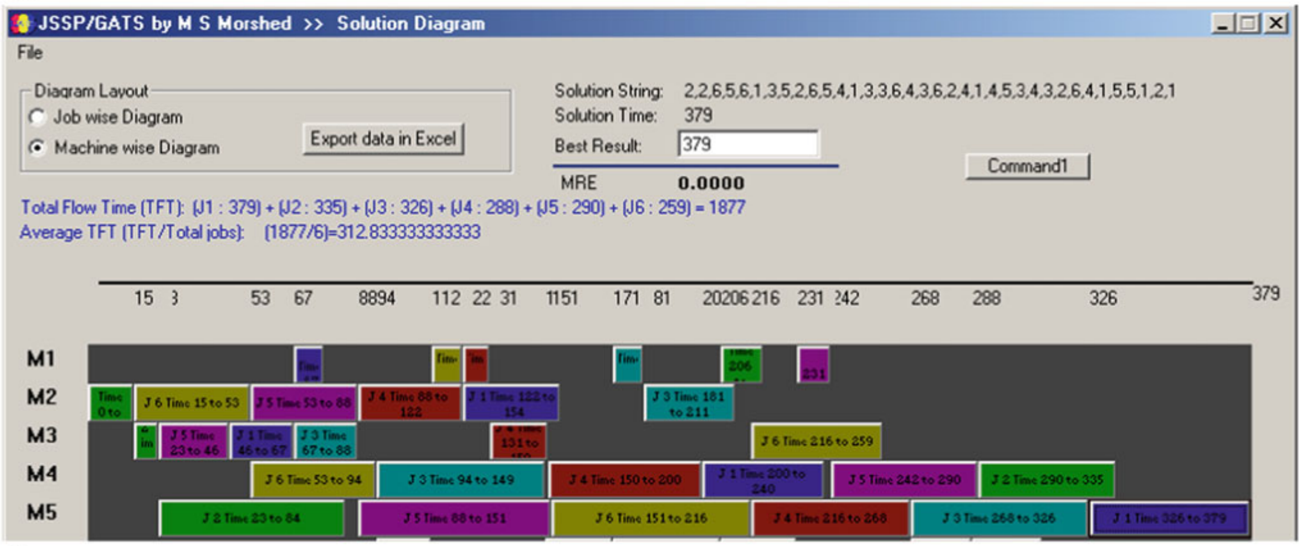
Total Flow Time (TFT): (J1 : 444) + (J2 : 336) + (J3 : 379) + (J4 : 353) + (J5 : 399) + (J6 : 291) = 2202
Average TFT (TFT/Total jobs): (2202/6)=367

Fig. 7 **a** Machine Gantt chart for JSSP2 test case. **b** Job Gantt chart for JSSP2 test case

Table 7 JSSP3 test case and results

J	O_1		O_2		O_3		O_4		O_5		O_6	
	M	P	M	P	M	P	M	P	M	P	M	P
1	3	21	1	10	2	32	4	40	6	23	5	53
2	2	15	3	8	5	61	6	35	1	14	4	45
3	3	21	4	55	6	22	1	10	2	30	5	58
4	2	34	1	9	3	19	4	50	5	52	6	20
5	3	23	2	35	5	63	6	25	1	11	4	48
6	2	38	4	41	6	18	1	10	5	65	3	43
Generation	Makespan (C_{max})	MRE %										
0	414 (lower) and 524 (upper)	8.45 and 27.67										
30	379	0.00										
60	379	0.00										
90	379	0.00										

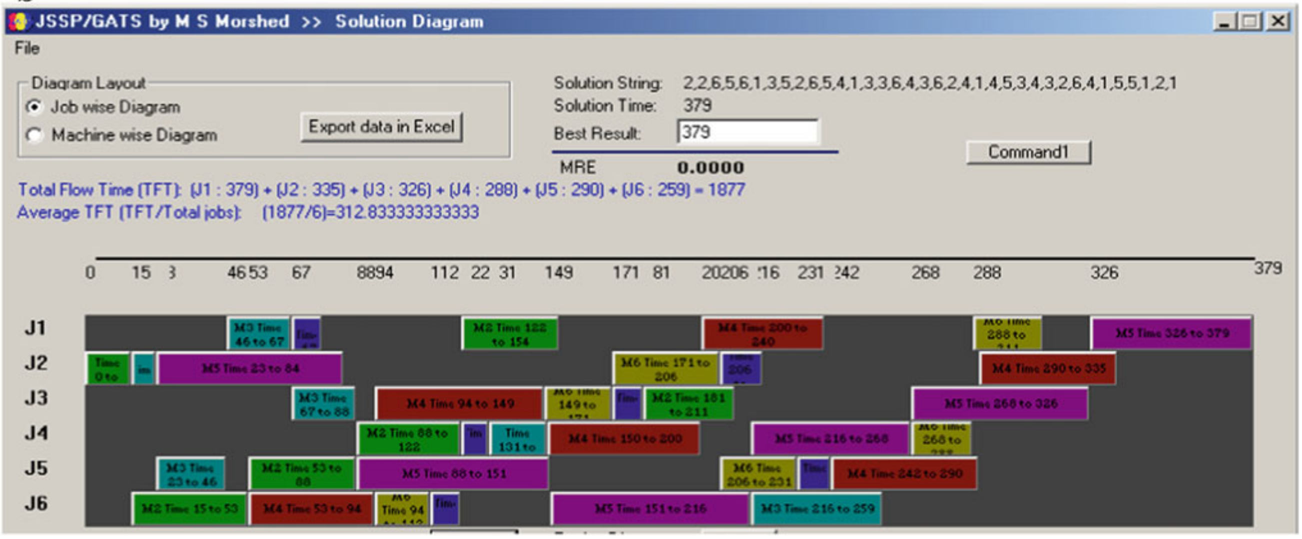
J = job, M = machine, P = processing time, O = operation a



Job wise Diagram [✓ Machine wise Diagram]

Solution String: 2,2,6,5,6,1,3,5,2,6,5,4,1,3,3,6,4,3,6,2,4,1,4,5,3,4,3,2,6,4,1,5,5,1,2,1
Solution Time: 379
Best Result: 379
MRE 0.0000
Total Flow Time (TFT): (J1 : 379) + (J2 : 335) + (J3 : 326) + (J4 : 288) + (J5 : 290) + (J6 : 259) = 1877
Average TFT (TFT/Total jobs): (1877/6)=312.833333333333

b



[✓ Job wise Diagram] Machine wise Diagram

Solution String: 2,2,6,5,6,1,3,5,2,6,5,4,1,3,3,6,4,3,6,2,4,1,4,5,3,4,3,2,6,4,1,5,5,1,2,1
Solution Time: 379
Best Result: 379
MRE 0.0000
Total Flow Time (TFT): (J1 : 379) + (J2 : 335) + (J3 : 326) + (J4 : 288) + (J5 : 290) + (J6 : 259) = 1877
Average TFT (TFT/Total jobs): (1877/6)=312.833333333333

Fig. 8 **a** Machine Gantt chart for JSSP3 test case with optimum result. **b** Job Gantt chart for JSSP3 test case with optimum result

The upper and lower bounds for this problem at the start of the search are 493 and 616 respectively with the minimum MRE value of 9.9%. However, the system presented is able to find a solution with make-span of 444 after the 10th generation for this problem. The current norm in the company is three and half days to complete these jobs. Hence the result provided by the proposed system is a substantial improvement.

Test case 3-JSSP3

Table 7 provides the problem data and results for the test case JSSP3. Figure. 8 shows the schedules obtained using the proposed system for JSSP3 in the form of Gantt charts.

This JSSP instance normally takes more than three eight hour shifts for the company to deliver the products. The system proposed here is able to obtain the improved minimum make-span of (379 units of time) within the 10th generation of execution. The initial MRE has been noted to be 8.45% for the make-span of 414. The company could reduce around 75% of its production time by just reducing the machine idle time through the implementation of the proposed system.

The make-span values which have been obtained for the test cases above are obtained with the following parameters; run 1–25, population size 30–200, generation 10–300, crossover rate 70–80%, mutation rate 1–10%, iterations for tabu search 20–100, restricted list 6–36. It can be seen that all the problems have the tendency to improve on the make-span value as the number of generations increases. In most of the

Table 8 Results for test cases in terms of number generations of the search

Problem number	MRE at starting	MRE at 30th generation	MRE at 60th generation	MRE at 90th generation	Best C_{max} obtained (min)
1	6.31 (lower) and 41.10 (upper)	0	0	0	505
2	9.94 (lower) and 27.92 (upper)	0	0	0	444
3	8.45 (lower) and 27.67 (upper)	0	0	0	379

cases within the 10 generations of hybrid search substantial improvement is achieved. In these cases the company's normal schedule took four times longer than the make-spans the proposed system achieved. These results indicate the applicability of the proposed system to real life scenarios. Hence this scheduling model can be adapted for industrial use as a generic model (Table 8).

Conclusion

Even though there are many hybrid systems developed recently, only a small number of systems appears to be using GA and TS together thereby exploiting the complementary strengths of global parallel search of GA and local optimum avoidance of TS for solving JSSP. The system presented here is one such system. In this system TS is directly used in solution string exploration (of GA) making the input format common to both GA and TS. The system also appears to be one of the few systems which can find a solution to any arbitrary job shop scheduling problem rather than being a problem specific method. This hybrid model exhibits a polynomial computational complexity due to the incorporation of new heuristic techniques such as the mid-cut heuristic. The system described here is able to find the optimal solutions or at least near optimal solutions for real life problems faced by companies such as the automobile windscreen producer and the steel mill. Moreover, when this system was tested on 51 benchmark problems that exist in the literature it found optimum solutions for 48 of these problems and achieved an average MRE of 0.08% details of which is given elsewhere in Meeran and Morshed (2007).

The proposed model has been used on different types of real-life practical problems. In almost all cases the proposed system performed better. However the comparison of the results obtained and presented above for scheduling three test job shops in the steel mill example was difficult as the existing scheduling methods in the company are very crude not based on any established standards. Most of the products the company produced in their job shops normally took at least four times longer to complete than what the system proposed has achieved. The worker and shop supervisor inserted large amount of idle time for processing any job as they do not have standard systems of scheduling. The supervisor instructed the worker to load the arrived jobs and start processing without exploring the possibility of maximising the utility of machines and minimising the overall scheduling length.

As shown above, this model has been used the real life job shop environment in a real company and has provided a breakthrough for the company in reducing its schedule times. On all the job shop cases on which this framework has been tested improved results have been achieved. The computational time for each case ranged from 0.10 s to 48 min depending on the model parameters and problem size. The proposed model has been applied on these real life combinatorial job shop scheduling problems without any modifications, which proves the versatility of the proposed system and its nature of being a problem independent system.

Acknowledgments Both the authors would like to thank Prof. Ball, then Head of Mechanical and Manufacturing Engineering, University of Birmingham for allowing the first author to provide partial funding to the second author for his PhD which formed the basis of this paper. The first author also would like to thank Prof. Paul Goodwin, School of Management, University of Bath and Fatima Sheik, University of Birmingham for their help in proof reading this paper.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job-shop scheduling. *Management Science, 34*(3), 391–401.

Aydin, M. E., & Fogarty, T. C. (2004). A simulated annealing algorithm for multi-agent systems: A job-shop scheduling application. *Journal of Intelligent Manufacturing, 15*(6), 805–814.

Barnes, J. W., & Chambers, J. B. (1995). Solving the job shop scheduling problem using tabu search. *IIE Transactions, 27*, 257–263.

Blazewicz, J., et al. (1996). *Job shop scheduling, scheduling computer and manufacturing processes* (pp. 16–20 and 265–317). Germany: Springer.

Chen, L., Bostel, N., Dejax, P., Cai, J., & xi, L. (2007). A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal. *European Journal of Operational Research, 181*(1), 40–58.

Chen, C. L., & Chen, C. L. (2009). Bottleneck-based heuristics to minimize total tardiness for the flexible flow line with unrelated parallel machines. *Computers and Industrial Engineering, 56*(4), 1393–1401.

Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms-I. *Representation, Computers & Industrial Engineering, 30*(4), 983–997.

Chiang, T. C., & Fu, L. C. (2007). Using dispatching rules for job shop scheduling with due-date based objectives. *International Journal of Production Research, 45*(14), 3245–3262.

Chiu H. P., Hsieh, K. L., Tang, Y. T., & Wang C. Y. (2007). A tabu genetic algorithm with search adaptation for the job shop scheduling problem. In *Proceedings of the 6th WSEAS international conference on artificial intelligence, knowledge engineering, data bases, Greece, February 16–19*.

Davis, L. (1985). Job-shop scheduling with genetic algorithm. In *Proceedings of the 1st international conference on genetic algorithms and their applications, Pittsburgh, PA* (pp. 136–140).

Dell'Amico, M., & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research, 41*, 231–252.

Della Croce, F., Tadei, R., & Volta, G. (1995). A genetic algorithm for the job shop problem. *Computers and Operations Research, 22*(1), 15–24.

Demirkol, E., Mehta, S., & Uzsoy, R. (1997). A computational study of shifting bottleneck procedures for shop scheduling problems. *Journal of Heuristics, 3*(2), 111–137.

Dorndorf, U., & Pesch, E. (1995). Evolution based learning in a job-shop scheduling environment. *Computers and Operations Research, 22*(1), 25–40.

Eswarmurthy, V., & Tmilarasi, A. (2009). Hybridizing tabu search with ant colony optimization for solving job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology, 40*, 1004–1015.

Fattahi, P., Mehrabad, M. S., & Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing, 18*(3), 331–342.

Gao, J., Gen, M., & Sun, L. (2006). Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *Journal of Intelligent Manufacturing, 17*(4), 493–507.

Gen, M., Gao, J., & Lin, L. (2009). Multistage-based genetic algorithm for flexible job-shop scheduling problem. *Intelligent and Evolutionary Systems, Studies in Computational Intelligence, 187*, 183–196.

Gen, M., Tsujimura, Y., & Kubota, E. (1994). Solving job-shop scheduling problems by genetic algorithm. In *Proceeding of 1994 IEEE international conference on systems, man, and cybernetics* (Vol. 2, pp. 1577–1582).

Gholami, M., & Zandieh, M. (2009). Integrating simulation and genetic algorithm to schedule a dynamic flexible job shop. *Journal of Intelligent Manufacturing, 20*(4), 481–498.

Glover, F. (1989). Tabu search—Part I. *ORSA Journal on Computing, 1*, 190–206.

Goncalves, J. F., Mendes, J. D., & Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research, 167*, 77–95.

González, M. A., Vela, C. R., & Varela, R. (2009). Genetic algorithm combined with tabu search for the job shop scheduling problem with setup times' methods and models in artificial and natural computation. A homage to professor Mira's scientific legacy. *Lecture Notes in Computer Science, 5601/2009*, 265–274. doi:10.1007/978-3-642-02264-7_28.

Jain, A. S., & Meeran, S. (1998a). Job-Shop Scheduling using neural networks. *Internation Journal of Production Research, 36*(5), 1249–1272.

Jain, A. S., & Meeran, S. (1998b). *A state-of-the-art review of job-shop scheduling techniques*. Technical Report, University of Dundee, UK.

Jain, A. S., & Meeran, S. (2002). A multi-level hybrid framework applied to the general flow-shop scheduling problem. *Computers and Operations Research, 29*, 1873–1901.

Jones, A., & Rabelo, L.C. (1998). *Survey of job shop scheduling techniques*. Gaithersburg, MD: National Institute of Standards and Technology (NISTIR).

Mascis, A., & Pacciarelli, D. (2002). Job shop scheduling with blockings and no-wait constraints. *European Journal of Operational Research, 143*, 498–517.

Mattfeld, D. C. (1996). *Evolutionary search and the job shop: Investigations on genetic algorithms for production scheduling*. Heidelberg, Germany: Physica-Verlag.

Meeran, S. & Morshed, M. S. (2007, December). A hybrid configuration for solving job shop scheduling problems. In *8th Asia Pacific industrial engineering and mangement science Conference, Kaohsiung, Taiwan*.

Morshed, M. S. (2006). *A hybrid model for job shop scheduling*. PhD Thesis, University of Birmingham, UK.

Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job-shop problem. *Management Science, 42*(6), 797–813.

Panwalkar, S. S., & Iskander, W. (1977). A survey of scheduling rules. *Operations Research, 25*(1), 45–61.

Park, B. J., Choi, H. R., & Kim, H. S. (2003). A hybrid genetic algorithm for the job shop scheduling problems. *Computers & Industrial Engineering, 45*, 597–613.

Pérez, E., Posada, M., & Herrera, F. (2010). Analysis of new niching genetic algorithms for finding multiple solutions in the job shop scheduling. *Journal of Intelligent Manufacturing*, Online Firsttrademark, March 10, 2010.

Pezzella, F., & Merelli, E. (2000). A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operation Research, 120*, 297–310.

Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers and Operations Research, 35*, 3202–3212.

Pinedo, M., & Chao, X. (1999). *Operations scheduling with applications in manufacturing and services*. Singapore: McGraw Hill.

Rossi, A., & Boschi, E. (2009). A hybrid heuristic to solve the parallel machines job-shop scheduling problem. *Advances in Engineering Software, 40*(2), 118–127.

Roy, B., & Sussmann, B. (1964). *Les Problemes d'Ordonnancement avec Contraintes Disjonctives*. Note D.S. no. 9 bis, SEMA, Paris, France, December 1964.

Satake, T., Morikawa, K., Takahashi, K., & Nakamura, N. (1999). Simulated annealing approach for minimizing the make-span of the general job shop. *International Journal of Production Economics, 60*(61), 515–522.

Suresh, R. K., & Mohanasundaram, K. M. (2005). Pareto archived simulated annealing for job shop scheduling with multiple objectives. *The International Journal of Advanced Manufacturing Technology, 29*, 184–196.

Tamilselvan, R., & Balasubramanie, P. (2009). Integrating genetic algorithm, tabu search approach for job shop scheduling. *International Journal of Computer Science and Information Security, 2*(1).

Tan, Y., Liu, S.,& Wang, D. (2010). A constraint programming-based branch and bound algorithm for job shop problems. In *Control and decision conference (CCDC), 2010 China, May 26–28, 2010* (pp. 173–178).

Tay, J. C., & Ho, N. B. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering, 54*(3), 453–473.

Thomsen, S. (1997). *Meta-heuristics combined with branch & bound*. Technical Report. Copenhagen Business School, Copenhagen, Denmark.

Wang, L., & Zheng, D. Z. (2001). An effective hybrid optimisation strategy for job shop scheduling problems. *Computers and Operations Research, 28*, 585–596.

Weckman, G. R., Ganduri, C. V., & Koonce, D. A. (2008). A neural network job-shop scheduler. *Journal of Intelligent Manufacturing, 19*(2), 191–201.

Yamada, T. & Nakano, R. (1996). Scheduling by genetic local search with multi-step crossover. In *PPSN'IV 4th international conference on parallel problem solving from nature, Berlin, Germany* (pp. 960–969).

Yu, H., & Liang, W. (2001). Neural network and genetic algorithm-based hybrid approach to expanded job shop scheduling. *Computers and Industrial Engineers, 39*, 337–356.

Zhang, G., Gao, L., & Shi, Y. (2010). A genetic algorithm and tabu search for multi objective flexible job shop scheduling problems (CCIE). In *2010 International conference on computing, control and industrial engineering, June 5–6* (pp. 251–254).

Zhang, H., & Gen, M. (2009). A parallel hybrid ant colony optimisation approach for job-shop scheduling problem. *International Journal of Manufacturing Technology and Management, 16*(1–2), 22–41.

Zhang, R., & Wu, C. (2008). A hybrid approach to large-scale job shop scheduling. *Applied Intelligence, 32*(1), 47–59.

Zhou, R., Nee, A. Y. C., & Lee, H. P. (2009). Performance of an ant colony optimisation algorithm in dynamic job shop scheduling problems. *International Journal of Production Research, 47*, 2903–2920.