A Decision Support System for Solving Job-Shop Scheduling Problems using Genetic Al-gorithms

Publication details:
Asia-Pacific Symposium on Intelligent & Evolutionary Systems
pp. 71-79
9780646506715 (ISBN)

Event details:
The 12th Asia-Pacific Symposium on Intelligent & Evolutionary Systems (IES'08)
Melbourne, Australia

Publication Date:
2008

DOI:
https://doi.org/10.26190/unsworks/544

A Decision Support System for Solving Job-Shop Scheduling Problems Using Genetic Algorithms

S. M. Kamrul Hasan, Ruhul Sarker and Daryl Essam

School of Information Technology and Electrical Engineering
University of New South Wales at the Australian Defence Force Academy
Northcott Drive, Canberra ACT 2600, Australia
Email: {kamrul, r.sarker, d.essam}@adfa.edu.au

Abstract: The primary objective of this research is to solve the job-shop scheduling problems by minimizing the makespan. In this paper, we first developed a genetic algorithm (GA) for solving JSSPs, and then improved the algorithm by integrating it with three priority rules. The performance of the developed algorithm was tested by solving 40 benchmark problems and comparing their results with that of a number of well-known algorithms. For convenience of implementation, we developed a decision support system (DSS). In the DSS, we built a graphical user interface (GUI) for user friendly data inputs, model choices, and output generation. An overview of the DSS and the analysis of experimental results are provided.

Keywords: Job-shop scheduling, genetic algorithm, decision support system, priority rule, makespan,

1. Introduction

The job-shop scheduling problem (JSSP) is one of the well known practical planning problems in the area of manufacturing and production planning. A classical JSSP is a combination of *N* jobs and *M* machines. Each job is composed of a set of operations that has to be processed, on a set of known machines, and where each operation has a known processing time. A schedule is a complete set of operations, required by a job, to be performed on different machines, in a given order. In addition, the process may need to satisfy other constraints such as (i) no more than one operation of any job can be executed simultaneously and (ii) no machine can process more than one operation at the same time. The widely used objectives considered in JSSPs are the minimization of makespan, the minimization of tardiness, and the maximization of throughput. Makespan is the total time between the starting of the first operation and the ending of the last operation. In this research, the objective considered is the makespan minimisation as this is more practical than other objectives used in solving JSSPs [1-7].

The JSSP is commonly known as one of the most difficult NP-complete problems [8] and is also well-known for its practical applications in many manufacturing industries. Over the last few decades, many algorithms have been developed to solve JSSPs. However, no single algorithm is suitable for solving all kinds of JSSP with both a reasonably good solution and within a reasonable computational effort. Thus, there is scope to analyze the difficulties of JSSPs as well as to design improved algorithms that may be able to solve them effectively.

Over the last few decades, a large amount of research has been reported that aim to solve JSSP by using Genetic Algorithms (GAs) and hybrid GAs [3, 5, 9-12]. The earliest application of GAs for solving JSSPs was reported in the mid-80s by Lawrence [13]. In recent times, it is a common practice to improve the performance of GA by incorporating different search and heuristic techniques, and this approach is readily applied to solving the JSSPs. For example, the hybrid methods proposed by Shigenobu *et al.* [14], Park *et al.* [15], Della Croce *et al.* [10] and Ombuki and Ventresca [11].

Although the development of efficient algorithms for solving any complex problem is very important, an appropriate implementation process of these algorithms is equally important. As the user and planner of job-shop scheduling may not be expert on optimization, computer programming and genetic algorithms, the development of a decision support system (DSS) would help in implementing the algorithm without understanding (or going through) the complex methodology involved.

DSS is a computer-based interactive system that supports decision makers utilizing data and models. It solves problems with various degrees of structures and focuses on the effectiveness rather than the efficiency of the decision process [16]. The computer programs need to be interactive with options to change all the parameters. Also, it needs to carry out all the detailed information about scheduling. For problems like JSSPs, a Gantt chart is preferable to represent the solutions graphically [17].

Numerous works have appeared on the development of DSS for different versions of scheduling problems, especially industrial scheduling problems or JSSPs. In the early 1980s, Viviers [17] developed a DSS to solve JSSPs. As well as their interactive user interface, they closely focused on the management issues related to decision making, such as those of: accepting or rejecting orders, subcontracting, increasing capacity or workload, breaking down the jobs, assigning jobs to artesans, assigning due-date and reassigning the jobs if necessary. In their model based management system, their objectives were to minimize the work-in-progress, as well as

reducing the lead time. Moreover, improving customer satisfaction by attaining due-dates was considered. More recently, Speranza and Werlee [18] worked to link the fields of DSS and Operations Research (OR), and discussed how DSS can take advantage of the methodology of OR. Hence they consequently focused on scheduling problems, as one of the most challenging applications where both DSS and OR can be applied successfully. The authors raised certain issues to justify the effectiveness of DSS over simplified models, such as: dynamic decision making, which may change depending on the particular situation, the inability to impose the entire knowledge of a decision maker in the models, the presence of unusual circumstances like political issues, the knowledge of users over the problems and the preparation of complex data. These issues are common in real life problems while they are hard to include in the models. In their models, they considered flexibility for users, including; reassigning due date and operation priorities, resetting total number of products to be produced, and moving an operation to a particular machine. They have reported that the model is capable of adjusting overlapping conditions and prohibits invalid machine selection. Also recently, McKay and Buzacott [19] and McKay and Wiers [20] developed a computerized system for solving scheduling problems, more precisely: JSSPs and timetabling. The first work may not be treated as DSS due to the absence of some necessary DSS components such as data base management. The authors implemented an interactive computer based interface for solving JSSPs. In their later work, they emphasized the decision making functionalities, such as start of day routine and special periods (like Friday, last day of the month etc.) for timetabling, and categorized their system as a DSS. Petrovic *et al.* [21] developed a decision support tool for solving JSSPs that used a fuzzy-genetic model. Their main emphasis was also on model based management. They considered multi-objective GA as a model for problem solving. Our main focus is also on the problem solving methodology and its relationship with the intractability of the DSS. At the current stage, we considered benchmark problems, rather than real-life industrial problems, to better judge the system performance. Silva *et al.* [22] developed a DSS for use in the mould industry, for their production planning. They mainly combined a system model, data model, and MAPP (Mould: Assistant Production Planner) to form the system. Data coming from the client are processed and stored in a DBMS, which is used by the application server. A web based client module is used to interface with the system. In the work of Kumar and Rajotia [23], they have integrated the process plan generator, DBMS and the scheduler with the DSS. The job-scheduling operations are performed by the scheduler, where the process plan generator organizes different tools for generating an appropriate machine setup. DSS can more generally be applied on numerous applications, including in scheduling, as may be found in the survey of Eom and Lee [16] and [24].

In this research, we first develop a traditional genetic algorithm (TGA) for solving JSSPs which includes only the basic components of an evolutionary algorithm. In this GA, each individual represents a particular schedule and the individuals are represented by a sequence of binary numbers which is commonly known as a chromosome. The chromosomes evolve in every generation by changing the arrangement of the binary bits. After reproduction, each and every infeasible individual is repaired to be feasible. This can also be termed as a genotype representation. On the other hand, the phenotype representation of the problem is a matrix of $m \times n$ integer numbers, where each row represents the sequence of jobs for a given machine. We mostly focused on the phenotype representations to analyze the schedules. The binary genotype is effective for the simple crossover and mutation techniques.

We then improve the TGA solutions by incorporating three priority rules, namely: partial reordering (PR), gap reduction (GR) and restricted swapping (RS). The details of these priority rules are discussed in a later section and also in [25, 26]. To test the performance of our proposed algorithms, we have solved 40 benchmark problems originally presented in Lawrence [27]. Our algorithm is able to obtain the exact optimal solutions for 27 out of 40 test problems. The overall performance of our algorithm is better than many of the key JSSP algorithms appearing in the literature.

After successful implementation of the algorithms, we develop a graphical user interface (GUI), as a part of the decision support system to give the user better flexibility in choosing parameters, in selecting appropriate algorithms, and in generating the desired outputs. We incorporate the decision making facilities for better management facilities. The output has the option to visualize the schedule in Gantt chart form. Regarding the algorithmic viewpoint, the current version of our algorithms is a modified but improved version from our earlier publications. The earlier version of these algorithms with experimental results on fewer test problems can be found in Hasan *et al.* [25, 26, 28].

The paper is organized as follows. After the introduction, the problem definition is presented in Section 2. Section 3 discusses the chromosome representation for JSSPs, and how to handle infeasibility in JSSPs. Section 4 introduces new priority rules for improving the performance of traditional GA. Section 5 presents the development of the decision support system including GUI. Section 6 presents the development of our proposed algorithms and implementation aspects. Section 7 provides both experimental results and parameter analysis. Finally, the conclusions and future research directions are presented.

2. Definition of a Standard JSSP

The standard job-shop scheduling problem makes the following assumptions:

- Each job consists of a finite number of operations.
- The processing time for each operation using a particular machine is defined.
- There is a pre-defined sequence of operations that has to be maintained to complete each job.
- Delivery times of the products are undefined.
- There is no setup or tardiness cost.
- A machine can process only one job at a time.
- Each job is performed on each machine only once.
- No machine can deal with more than one type of task.
- The system cannot be interrupted until each operation of each job is finished.

- No machine can halt a job and start another job before finishing the previous one.
- Each and every machine has full efficiency.

The objective of the problem is the minimization of the total time taken to complete each and every operation, while satisfying the machining constraints and required operational sequence of each job. In this research, we develop three different algorithms for solving JSSPs. These algorithms are briefly discussed in the next three sections.

3. Job-Shop Scheduling with GA

As indicated earlier, we consider the minimization of makespan as the objective of JSSPs. According to the problem definition, the sequence of machine use (this is also the sequence of operations as any one machine is capable of performing only one type of operation) by each job is given. In this case, if we know either the starting or finishing time of each operation, we can calculate the makespan for each job and hence generate the whole schedule. In JSSPs, the main problem is to find the sequence of jobs to be operated on each machine that minimizes the overall makespan. The chromosome representation is an important issue in solving JSSPs using GAs.

In solving JSSPs using GAs, the chromosome of each individual usually comprises the schedule. Chromosomes can be represented by binary, integer or real numbers. Some popular representations for solving JSSPs are: operation based, job based, preference-list based, priority-rule based, and job pair-relationship based representations [29]. We select the job pair-relationship based representation for the genotype, as in [3-5, 30], due to the flexibility of applying genetic operators to it. In this representation, a chromosome is symbolized by a binary string, where each bit stands for the order of a job pair (u,v) for a particular machine m. This means that for an individual p, the job u must precede the job v in machine m. The job having the maximum number of 1s is the highest priority job for that machine. The binary string acts as the genotype of individuals. It is possible to construct a phenotype which is the job sequence for each machine. The binary representation is helpful if the conventional crossover and mutation techniques are used. We use the binary representation for the flexibility of applying simple reproduction operators. We also use the constructed phenotype as the chromosome on which to apply priority rules.

In this algorithm, we perform simple two-point crossover and bit flip mutation. The crossover points are selected randomly. After performing crossover and mutation, we map the phenotype directly from the binary string i.e. the chromosome. As the reproduction operations are applied on a random basis, it does not ensure feasibility. This is why we apply two repairing techniques: local and global harmonization, in order to make the infeasible solutions into feasible solutions. Local harmonization is used during construction of the phenotype (i.e. the sequence of operations for each machine) from the binary genotype. From a chromosome of length l, m tables are formed. The technique was also fused in [3, 5, 31].

Global harmonization is a repairing technique applied directly on the phenotype for migrating infeasible solutions into feasible solutions. For an $m \times n$ job-shop scheduling problem, there will be $(n!)^m$ possible solutions. Only a small percentage of these solutions are feasible. The solutions mapped from the chromosome do not guarantee feasibility. Global harmonization swaps between the operations to reach the nearest feasible solution.

Suppose job j_3 specifies its first, second and third operations are to be processed on machines m_3, m_2 and m_1 respectively, and the job j_1 specifies its first, second and third operations on machines m_1, m_3 and m_2 respectively. Further assume that an individual solution (or chromosome) indicates that j_3 is scheduled on machine m_1 first as its first operation, followed by job j_1. Such a schedule is infeasible as it violates the defined sequence of operations for job j_3. In this case, the swapping of places between job j_1 and job j_3 on machine m_1, would allow job j_1 to have its first operation on m_1 as required, and it may provide an opportunity for job j_3 to visit m_3 and m_2 before visiting m_1 as per its order. Usually, the process identifies the violations sequentially and performs the swap one by one until the entire schedule is feasible. In this case, there is a possibility that some swaps performed earlier in the process are required to swap back to their original position to make the entire schedule feasible. This technique is useful not only for the binary representations, but also for the job-based or operation based representation. A detailed explanation of the local and global harmonization techniques is given in our earlier publications [25, 26].

In our proposed algorithm, we consider multiple repairs to narrow down the deadlock frequency. As soon as a deadlock occurs, the algorithm identifies at most one operation from each job that can be scheduled immediately. Starting from the first operation, the algorithm identifies the corresponding machine of the operation and swaps the tasks in that machine so that at least the selected task disallows deadlock for the next time. For n jobs, the risk of getting into deadlock will be removed for at least n operations.

After performing global harmonization, we obtain a population of feasible solutions. We then calculate the makespan of all the feasible individuals and rank them based on their fitness values. We then apply genetic operators to generate the next population. We continue this process until it satisfies the stopping criteria.

4. Priority Rules for GAs

As reported in the literature, different priority rules are imposed in conjunction with GAs to improve the JSSP solution. Dorndorf and Pesch [12] proposed twelve different priority rules for achieving better solutions for JSSPs. However they suggested choosing only one of these rules while evaluating the chromosome. In this section, we introduce three new priority rules. We propose using these rules on selected individuals after the fitness evaluation. The action of the rules will be accepted if and only if it improves the solution. As the improvement is passed to the chromosomes, which can be transferred to the offspring, it follows Lamarckian type learning [32]. These priority rules can be used as local/neighborhood search heuristics in conjunction with the GAs to improve the quality of solutions generated by GAs. The rules are briefly discussed below.

4.1. Partial Reordering (PR)

In the first rule, we identify the machine (m_k) which is the deciding factor for the makespan in phenotype p and the last job (j_k) that is to be processed by the machine m_k. The machine m_k can be termed as the bottleneck machine in the chromosome under consideration. Then we find the machine (say m') required by the first operation of job j_k. The reordering rule then suggests that the first operation of job j_k must be the first task on machine m' if it is not already scheduled.

4.2. Gap Reduction (GR)

After each generation, the generated phenotype usually leaves some gaps between the jobs. Sometimes, these gaps are necessary to satisfy the precedence constraints. However, in some cases, a gap could be removed or reduced by placing in it a job from the right side of the gap. For a given machine, this is like swapping between a gap from the left and a job from the right of a schedule. In addition, a gap may be removed or reduced by simply moving a job to its adjacent gap at the left. This process would help to develop a compact schedule from the left and continuing up to the last job for each machine.

4.3. Restricted Swapping (RS)

For a given machine, the restricted swapping rule allows swapping between adjacent jobs if and only if the resulting schedule is feasible. This process is carried out only for the job which takes the longest time for completion.

Suppose job j' takes the longest time for completion as the phenotype p. The algorithm starts from the last operation of j' in p and checks with the immediate predecessor operation whether these two are swappable or not. The necessary conditions for swapping are: none of the operations can start before the finishing time of the immediate predecessor operation of that corresponding job, and both operations have to be finished before starting the immediate successive operations of the corresponding jobs. More explanation of these rules with necessary figures are available in our previous publications [25, 26].

5. Development of the DSS

According to the definition, it is a simple job-shop scheduling problem that follows the problem definition mentioned in section 2. We have developed a standard decision support system to evaluate the management decisions and to execute the appropriate algorithm to solve simple JSSPs. Standard DSSs contain three basic subsystems: data management, model management and dialog management [33]. These three components usually interact between each other. This interaction consists of sharing resources, exchanging information and messages, passing feedback etc. Moreover, the whole management system interacts with the user interface (UI) to process the input and simulate the output. The flow diagram is presented in Fig. 1.

5.1. Data Base Management Subsystem (DBMS)

The DBMS is mainly the input processing subsystem. The major tasks of this subsystem are to reshape and simplify the incoming data. It handles

- the problem description i.e. sequence of operations and corresponding execution time
- reproduction parameters i.e. crossover and mutation probabilities, along with other selection parameters
- stopping criteria i.e. maximum allowable number of generations, particular delivery time or specific makespan to stop the iterative process.

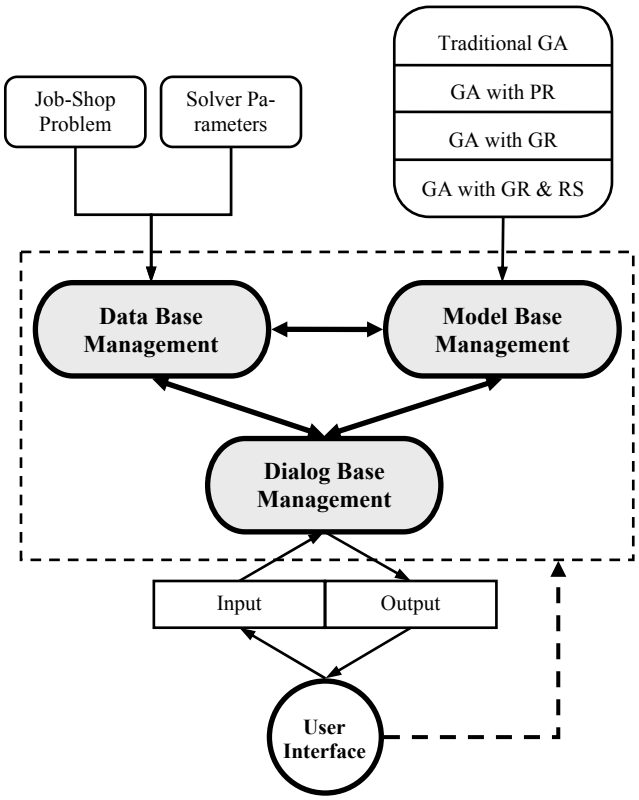

Fig. 1. Flow diagram of the decision support system.

The purposes of incoming data are elaborated in section 5.4.2. The DGMS shares the processed inputs with the model management to execute a selected model successfully. It also handles further requests of input data by any other units. On the other hand, it is connected to UI through DGMS, in terms of acquiring the inputs from the interactive user interface. Users load necessary data by using the UI which is finally captured by the DBMS.

5.2. Model Base Management Subsystem (MBMS)

MBMS deals with the different kinds of algorithms which can be treated as models. We have developed four different algorithms, which act as four different models. The MBMS controls the operations of those models. Moreover, it facilitates the models by ensuring appropriate support from other units.

It also keeps the process input data from the DBMS. The models use this data to execute themselves and to generate

the expected outputs. The MBMS needs the support of the UI to choose any particular model. Thus the user/manager has the access to select any particular model. In a sense, this unit is fully controllable from the UI.

5.3. Dialog Management Subsystem (DGMS)

The dialog component of a DSS is the hardware and software that provides the service of connectivity, between the user interface and other management systems. It also accommodates the user with a variety of input devices and stores input/output data. Sometimes the UI is also treated as a part of the DGMS [34]. According to Fig. 1, the UI is virtually connected to all three management subsystems, as it is directly linked and controlled by the DGMS. It handles the accessibility of the dialogs in the user interface, and maintains the input and output data flows between the UI and the other subsystems.

5.4. User Interface (GUI)

The UI is the main graphical component to control the interaction between the DSS and the end user or manager. It communicates directly with the DGMS and exchanges input and output data.

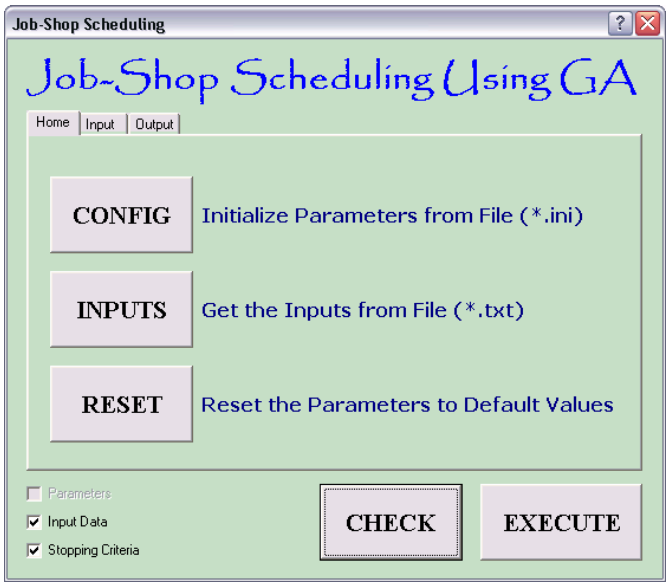

Fig. 2. Graphical user interface of the simple input options in the decision support system.

As we implemented three different rules and developed three different algorithms using those, we also considered the option to select any of the algorithms and traditional GA as well. For better convenience, we used an option to select the algorithm parameters from an initialization file.

Fig. 1. represents the flow diagram of the complete system. The decision support system takes the input data in three different classes, through the help of the DGMS and then passes it to the DBMS. The DGMS finally process the output and gives feedback to the user/manager. The output is generated in the form of values, as well as a Gantt chart.

5.4.1 Home

This option is for getting the simplified input parameters directly from the configuration file. All of the necessary algorithm parameters can be stored in files using a specific input format for the purposes of quickly selecting parameters. This tab contains two input buttons. The CONFIG button loads *.ini files to initialize the algorithm parameters, while the Inputs button loads the input files in *.txt format for operational sequences etc. The parameters can be reset to the default values using the RESET button.

5.4.2 Input

This option allows users to choose the input parameters, including the algorithm and stopping criteria, interactively from the interface itself. More precisely, it gives a clear graphical view of each and every component of the algorithms.

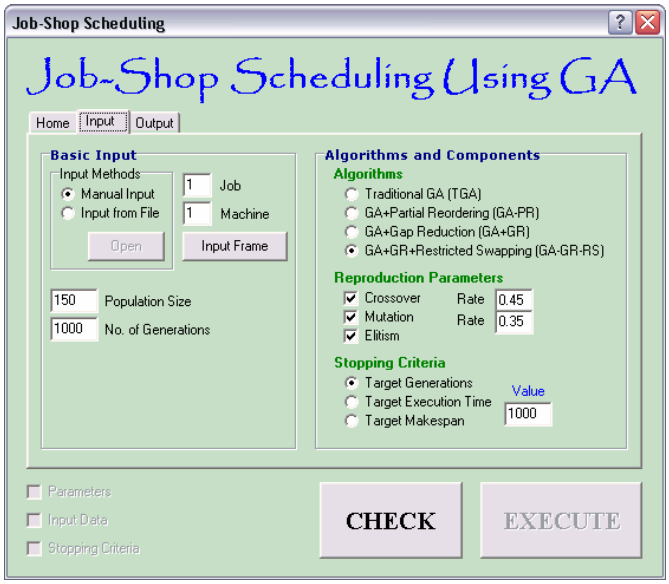

Fig. 3. Graphical user interface of the advanced input in the decision support system.

The manual input allows a user to insert all of the parameters of the algorithms manually where smaller amounts of data have been used. This is helpful in the case of small scale problems. Selecting manual input enables the edit box to specify the total number of machines and jobs. On the other hand, the input data can be loaded from a file, which is much more convenient for large problems. As we initially implemented a traditional GA, and then later applied the priority rules, we kept the option for all four of the algorithms, including TGA. The two main parameters are the size of population and total number of generations. The value of the number of generations is the default stopping criteria in any critical circumstances.

The stopping criterion is essential for better management. As it allows a manager to specify the criteria to stop the iterative process. The program may stop after a certain number of generations, or after a particular period of time, or after achieving a specific makespan. In the case of makespan, if it is not achieved within the maximum number of generations,

the program stops. This is to avoid infinitely approaching towards an infeasible makespan. There is also an option to insert the reproduction parameters i.e. crossover and mutation probability, while the elitism technique is to keep the best solution of every generation unchanged in the next generation. Adding elitism ensures the continuation of the best solution in every following generation.

5.4.3 Output

The simplified output screen shows the necessary information to measure the quality of the solutions, as well as a graphical view of the best solution in the form of a Gantt chart. The computational results contain: the best makespan found, which the solution with the minimum completion time is; the average makespan of all the solutions; and the worst fitness, in the scale of unit time. Moreover, it also includes: the standard deviation of all the solutions, total execution time to reach to the current solution, total number of fitness evaluations, and the cumulative idle time between each consecutive operation.

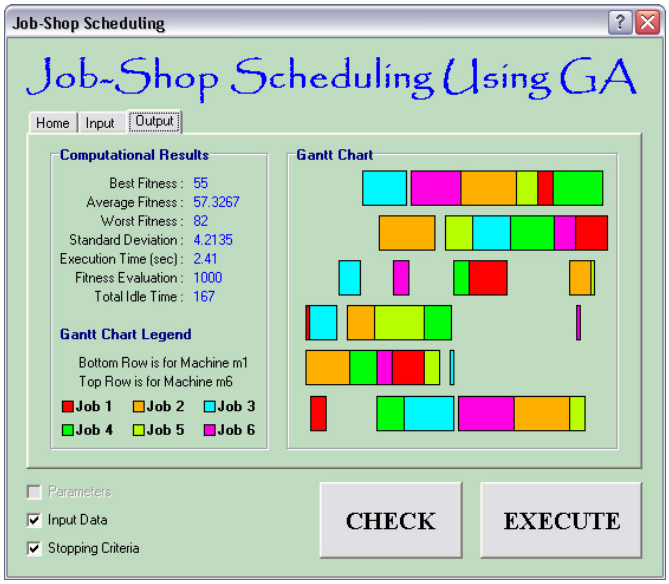

Fig. 4. Graphical user interface of the output in the decision support system.

In the Gantt chart, each color represents the operations of a particular job which is mentioned in the legend. The chart contains M rows, where M is the total number of machines. Each row contains N different operations represented by N colors, where N is the total number of jobs. Empty spaces between each pair of operations are the idle time for that particular machine. The Gantt chart is the best and most effective way to visualize the schedules in the time domain.

6. Implementation of Genetic Algorithms

First, we implemented a simple GA for solving JSSPs. Each individual is represented by a binary chromosome. We use the job-pair relationship based representation as of Nakano and Yamada [3] and Paredis *et al.* [35]. We use simple two point crossover and bit flip mutation as reproduction operators. We carried out a set of experiments with different crossover and mutation rates to analyze the performance of the algorithm. After the successful implementation of the GA, we implemented three versions of GAs by introducing the priority rules, as discussed in the last section, as follows:

- GA-PR: Partial re-ordering rule with GA
- GA-GR: Gap reduction rule with GA and
- GA-GR-RS: Gap reduction and restricted swapping rule with GA

In all versions of GA, if selected, we apply elitism in each generation to preserve the best solution found so far, and also to inherit the elite individuals more than the rest [36, 37]. In performing the crossover operation, we use the tournament selection that chooses one individual from the elite class of the individuals (i.e. the top 15%) and two individuals from the rest. This selection then plays a tournament between the last two and performs crossover between the winner and the elite individual. We rank the individuals on the basis of their fitness value. A high selection pressure on the better individuals may contribute to premature convergence. In particular, we consider the situation where 50% or more of the elite class are the same solution. In this case, their offspring will be quite similar after some generations. To counter this, when this occurs, a higher mutation rate will be used to help to diversify the population. We varied the crossover and mutation rate and made several experiments. Based on the result, the optimum rate found was 0.45 for crossover and 0.35 for mutation. Detailed results and analysis is listed in the next section. We set the population size to 2500 and the number of generations to 1000. Note that JSSPs usually require a high population size. For example, [38] used a population size of 5000 even for 10×10 problems. In our approach, when chosen, GR is applied to every individual. On the other hand, we apply PR and RS to only 5% of randomly selected individuals in every generation. To test the performance of our proposed algorithms, we have solved the 40 benchmark problems designed by Lawrence [27] and have compared our results with several existing algorithms. The problems range from 10×5 to 30×10 where n×m represents n jobs and m machines.

7. Result and Analysis for GAs

The results for the benchmark problems were obtained by executing the algorithms on a personal computer. Each problem was run 30 times and the simplified results are tabulated in Table 1.

Table 1. Comparing Our Four Algorithms

No. of Problems	Algorithm	Optimal Found	ARD (%)	SDRD (%)	Fitness Eval. (10^3)
40 (la01–la40)	TGA	15	3.591	4.165	664.90
	PR-GA	16	3.503	4.192	660.86
	GR-GA	23	1.360	2.250	356.41
	GR-RS-GA	27	0.968	1.656	388.58

To analyze the individual contribution of the priority rules, we experiment on a sample of five problems (la21-la25) with the same set of parameters in the same computing environ-

ment. For these problems, the individual percentage improvements of PR, GR and RS over GA after 100, 250 and 1000 generations was recorded. Although all three priority rules have a positive effect, GR's contribution is significantly higher than the other two rules and is consistent over many generations.

The table compares the performance of the four algorithms we implemented [GA, GA-PR, GA-GR, and GA-GR-RS] in terms of the % average relative deviation (ARD) from the best result published in the literature, the standard deviation of % relative deviation (SDRD), and the average number of fitness evaluations required. From Table 1, it is clear that the performance of the GAs with priority rules are better than the traditional GA, and GA-GR is better than both GA-PR and TGA. The addition of RS to GA-GR, which is known as GA-GR-RS, has clearly enhanced the performance of the algorithm. Out of the 40 test problems, both GA-GR and GA-GR-RS obtained exact optimal solutions for 23 problems. In addition, GA-GR-RS obtained optimal solutions for another 4 problems and substantially improved solutions for 10 other problems. In general, these two algorithms converged quickly, which can be seen from the average number of fitness evaluations.

Interestingly, in the case of GR, the average rate of improvement gradually decreases as the generation number increases (8.92% after 100 generations and 6.31% after 1000 generations). The reason for this, is that GR starts with a set of high quality initial solutions. Alternatively PR and RS have no significant effect on the solutions after the first evaluation, but GR gives 18.17% more improved solutions compared to GA. This is measured by the average improvement of the best makespan after the first evaluation without applying any other genetic operators.

To observe the contribution more closely, we recorded the improvement due to the individual rule in every generation in the first 100 generations. It was observed that GR consistently outperformed the other two rules. PR is effective only for the bottleneck jobs, whereas GR was applied to all individuals. The process of GR eventually makes most of the changes performed by PR over some (or many) generations. We identified a number of individuals where PR could make a positive contribution. We applied GR on those individuals to compare their relative contribution. For the five problems we considered over 1000 generations, we observed that GR made a 9.13% higher improvement than PR. It must be noted here that GR is able to make all the changes which PR does. That means PR cannot make an extra contribution over GR. As a result, the inclusion of PR with GR does not help to improve the performance of the algorithm. That is why we do not present other possible variants of GAs, such as GA-PR-RS and GA-GR-RS-PR.

Both PR and RS were applied to only 5% of the individuals. The role of RS is mainly to increase the diversity. A higher rate of PR and RS does not provide significant benefits either in terms of quality of solution or computational time. We experimented with varying the rate of PR and RS individually, for five selected problems, from 5% to 25%. We observed that the increase of the rate of applying PR and RS does not improve the quality of the solutions. Moreover, it takes extra time to converge.

Table 2. Comparing the Algorithms Based on Average Relative Deviations and Standard Deviation of Average Relative Deviations

Author	Algorithm	ARD(%)	SDRD(%)
Our Proposed	GR-RS-GA	0.97	1.66
Aarts *et al.*	GLS1	4.00	4.09
Aarts *et al.*	GLS2	2.05	2.53
Dorndorf & Pesche	PGA	1.75	2.20
Dorndorf & Pesche	SBGA (40)	1.25	1.72
Binato *et al.*	-	1.87	2.78
Adams *et al*	SB I	3.67	3.98

We have compared the performance of our best algorithm GA-GR-RS with other published algorithms based on the average of relative deviation (ARD) and the standard deviation of the relative deviations (SDRD) as presented in Table 2. Our GA-GR-RS clearly outperformed all the algorithms compared in the table.

8. Conclusion

Although JSSP is a very old and popular problem, there is still no algorithm that can assure the optimal solution for all test problems, specifically for larger problems in the literature. However, GAs are gaining popularity due to their effectiveness of solving optimization problems within a reasonable time period. In this paper, we have presented genetic algorithm based approaches to solve job-shop scheduling problems. After developing a traditional GA with different kinds of operations, we have designed and implemented three priority rules and three versions of genetic algorithms. All three genetic algorithms provided superior results than the traditional GA for JSSPs. We have solved 40 benchmark problems and have compared results with well-known algorithms appearing in the literature. Our genetic algorithm GA-GR-RS clearly outperforms all the algorithms considered in this paper. We have shown the ability to integrate the algorithms with the decision support system. The interactive presentation and decision making ability of the system gives more effectiveness to the work. Regarding the results, we have also provided a sensitivity analysis of parameters and have also experimented with different parameters and algorithms for analyzing their contributions. Although our algorithm is performing well, we feel that the algorithm requires further work to ensure consistent performance for a wide range of practical JSSPs. Moreover, it also needs the flexibility of adding new jobs and adjusting due dates which forward the work few more steps towards practical JSSPs. We intend to extend our research by introducing constraints such as, machine breakdown, dynamic job arrival, machine addition and removal, and due date restrictions, which will also be included in the DSS. Moreover, we would also like to test the performance of our algorithm on large scale problems. However, the new genetic algorithm is a significant contribution to the research of solving JSSPs.

Reference

[1] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science,* vol. 34, pp. 391-401, 1988.

[2] S. Binato, W. Hery, D. Loewenstern, and M. Resende, *A GRASP for Job Shop Scheduling*: Kluwer Academic Publishers, 2000.

[3] R. Nakano and T. Yamada, "Conventional genetic algorithm for job shop problems," in *Fourth Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, San Mateo, California, 1991, pp. 474-479.

[4] T. Yamada, "Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems," in *Department of Applied Mathematics and Physics*. vol. Doctor of Informatics Kyoto, Japan: Kyoto University, 2003, p. 120.

[5] T. Yamada and R. Nakano, "Genetic algorithms for job-shop scheduling problems," in *Modern Heuristic for Decision Support*, UNICOM seminar, London, 1997, pp. 67-81.

[6] W. Wang and P. Brunn, "An Effective Genetic Algorithm for Job Shop Scheduling," *Proceedings of the Institution of Mechanical Engineers -- Part B -- Engineering Manufacture,* vol. 214, pp. 293-300, 2000.

[7] F. Della Croce, R. Tadei, and G. Volta, "A genetic algorithm for the job shop problem," *Computers & Operations Research,* vol. 22, pp. 15-24, 1995.

[8] M. R. Garey, D. S. Johnson, and R. Sethi, "The Complexity of Flowshop and Jobshop Scheduling," *Mathematics of Operations Research,* vol. 1, pp. 117-129, 1976.

[9] J. E. Biegel and J. J. Davern, "Genetic algorithms and job shop scheduling," *Computers & Industrial Engineering,* vol. 19, pp. 81-91, 1990.

[10] F. D. Croce, R. Tadei, and G. Volta, "A genetic algorithm for the job shop problem," *Computers & Operations Research,* vol. 22, pp. 15-24, 1995.

[11] B. M. Ombuki and M. Ventresca, "Local Search Genetic Algorithms for the Job Shop Scheduling Problem," *Applied Intelligence,* vol. 21, pp. 99-109, 2004.

[12] U. Dorndorf and E. Pesch, "Evolution based learning in a job shop scheduling environment," *Computers & Operations Research,* vol. 22, pp. 25-40, 1995.

[13] D. Lawrence, "Job Shop Scheduling with Genetic Algorithms," in *First International Conference on Genetic Algorithms* Mahwah, NJ, USA: Lawrence Erlbaum Associates, Inc., 1985.

[14] K. Shigenobu, O. Isao, and Y. Masayuki, "An Efficient Genetic Algorithm for Job Shop Scheduling Problems," in *Proceedings of the 6th International Conference on Genetic Algorithms*: Morgan Kaufmann Publishers Inc., 1995.

[15] B. J. Park, H. R. Choi, and H. S. Kim, "A hybrid genetic algorithm for the job shop scheduling problems," *Computers & Industrial Engineering,* vol. 45, pp. 597-613, 2003.

[16] H. B. Eom and S. M. Lee, "A Survey of Decision Support System Applications (1971-April 1988)," *Interfaces,* vol. 20, pp. 65-79, 1990.

[17] F. Viviers, "A decision support system for job shop scheduling," *European Journal of Operational Research,* vol. 14, pp. 95-103, 1983.

[18] M. G. Speranza and A. P. Woerlee, "A decision support system for operational production scheduling," *European Journal of Operational Research,* vol. 55, pp. 329-343, 1991.

[19] K. N. McKay and J. A. Buzacott, "The application of computerized production control systems in job shop environments," *Computers in Industry,* vol. 42, pp. 79-97, 2000.

[20] K. N. McKay and V. C. S. Wiers, "Integrated decision support for planning, scheduling, and dispatching tasks in a focused factory," *Computers in Industry,* vol. 50, pp. 5-14, 2003.

[21] D. Petrovic, A. Duenas, and S. Petrovic, "Decision support tool for multi-objective job shop scheduling problems with linguistically quantified decision functions," *Decision Support Systems,* vol. 43, pp. 1527-1538, 2007.

[22] C. Silva, L. Roque, and A. Almeida, "MAPP - A web-based decision support system for the mould industry," *Decision Support Systems,* vol. 42, pp. 999-1014, 2006.

[23] M. Kumar and S. Rajotia, "Integration of process planning and scheduling in a job shop environment," *The International Journal of Advanced Manufacturing Technology,* vol. 28, pp. 109-116, 2006.

[24] S. B. Eom, S. M. Lee, E. B. Kim, and C. Somarajan, "A Survey of Decision Support System Applications (1988-1994)." vol. 49: Palgrave Macmillan Journals on behalf of the Operational Research Society, 1998, pp. 109-120.

[25] S. M. K. Hasan, R. Sarker, and D. Cornforth, "Modified Genetic Algorithm for Job-Shop Scheduling: A Gap-Utilization Technique," in *Evolutionary Computation, IEEE Congress on*, Singapore, 2007, pp. 3804-3811.

[26] S. M. K. Hasan, R. Sarker, and D. Cornforth, "GA with Priority Rules for Solving Job-Shop Scheduling Problems," in *IEEE World Congress on Computational Intelligence*, Hong Kong City, Hong Kong, 2008, pp. 1913-1920.

[27] S. Lawrence, "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques," Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania 1984.

[28] S. M. K. Hasan, R. Sarker, and D. Cornforth, "Hybrid Genetic Algorithm for Solving Job-Shop Scheduling Problem," in *Computer and Information Science, 6th IEEE/ACIS International Conference on*, Melbourne, Australia, 2007, pp. 519-524.

[29] S. G. Ponnambalam, P. Aravindan, and P. S. Rao, "Comparative Evaluation of Genetic Algorithms for Job-shop Scheduling," *Production Planning & Control,* vol. 12, pp. 560-674, 2001.

[30] J. Paredis, "Handbook of Evolutionary Computation," in *Parallel Problem Solving from Nature 2* Brussels, Belgium: Institute of Physics Publishing and Oxford University Press, 1992.

[31] T. Yamada and R. Nakano, "Job-Shop Scheduling," in *Genetic algorithms in engineering systems*. vol. 55, A. M. S. Zalzala and P. J. Fleming, Eds.: The Institution of Electrical Engineers, 1997, pp. 134-160.

[32] Y. S. Ong and A. J. Keane, "Meta-Lamarckian learning in memetic algorithms," *Evolutionary Computation, IEEE Transactions on,* vol. 8, pp. 99-110, 2004.

[33] E. Turban, *Decision support and business intelligence systems*, 8th ed. Upper Saddle River, N.J.: Pearson Prentice Hall, 2007.

[34] E. Turban, "Decision Support Systems: An Overview," in *Decision support and expert systems : management support systems*, 3rd ed, C. Stewart, Ed. New York: Macmillan, 1993, pp. 83-129.

[35] J. Paredis, T. Back, D. Fogel, and Z. Michalewicz, "Exploiting constraints as background knowledge for evolutionary algorithms," in *Handbook of Evolutionary Computation*: Institute, 1997, pp. G1.2:1-6.

[36] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *Systems, Man and Cybernetics, Part C, IEEE Transactions on,* vol. 28, pp. 392-403, 1998.

[37] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Reading, Mass: Addison-Wesley Pub. Co, 1989.

[38] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the Flexible Job-shop Scheduling Problem," *Computers & Operations Research,* vol. 35, pp. 3202-3212, 2008.