An Effective Hybrid Genetic Algorithm for Job Shop Scheduling Problem

Bin Cai*, Shilong Wang, and Haibo Hu

Abstract—The job shop scheduling problem (JSSP) is well known as one of the most difficult combinatorial optimization problems. This paper presents a hybrid genetic algorithm for the JSSP with the objective of minimizing makespan. The efficiency of the genetic algorithm is enhanced by integrating it with a local search method. The chromosome representation of the problem is based on operations. Schedules are constructed using a procedure that generates full active schedules. In each generation, a local search heuristic based on Nowicki and Smutnicki’s neighborhood is applied to improve the solutions. The approach is tested on a set of standard instances taken from the literature and compared with other approaches. The computation results validate the effectiveness of the proposed algorithm.

Keywords—Genetic algorithm, Job shop scheduling problem, Local search, Meta-heuristic algorithm

I. INTRODUCTION

The job shop scheduling problem (JSSP) is one of the most difficult problems in combinatorial optimization that has garnered considerable attention due to both its practical importance and its solution complexity. Efficient methods for solving the JSSP have significant effects on profitability and product quality. During the last three decades, many solution methods have been proposed to solve the JSSP. Those approaches can be divided into two categories: exact methods and approximation algorithms. Exact methods, such as branch and bound, linear programming and decomposition methods, guarantee global convergence and have been successful in solving small instances. In manufacturing systems, most scheduling problems are very complex in nature and very complicated to be solved by exact methods to obtain a global optimal schedule. For the big instances there is a need for approximation algorithms, which include priority dispatch, shifting bottleneck approach, local search, and heuristic methods. Recently, using a high-level strategy to guide other heuristics, known as meta-heuristics, led to better and more appreciated results in a relatively short period. Therefore, a number of meta-heuristics were proposed in literature for the JSSP such as genetic algorithm (GA) [1]-[4], simulated annealing (SA) [5], taboo search (TS) [6], greedy randomized adaptive search procedure (GRASP) [7] etc. A comprehensive survey of job shop scheduling techniques has been done by Jain and Meeran [8].

Among the meta-heuristic algorithms, GA has been used with increasing frequency to address scheduling problems. The GA is based on the survival of the fittest and involves some selection, crossover and mutation operations. GA exhibits parallelism, contains certain redundancy and historical information of past solutions, and is suitable for implementation on massively parallel architecture. As GA became popular in the mid 1980s, many researchers started to apply this meta-heuristic method to the JSSP. Yamada and Nakano [1] designed a GA for solving the classical JSSP. Kobayashi, Ono and Yamamura [4] designed another GA for the classic problem, and reached solution with high quality. Cheng, Gen and Tsujimura [9]-[10] provided a tutorial survey of works on solving the classical JSSP using GA. Wang and Zheng [11] developed a hybrid optimization strategy for JSSP. Ombuki and Ventresca [12] proposed a local search genetic algorithm to solve JSSP. Goncalves, Mendes and Resende [13] developed another hybrid genetic algorithm for JSSP.

In this paper, an effective hybrid intelligent algorithm for JSSP based on genetic algorithm and local search is presented. The remainder of the paper is organized as follows. An introduction for the job shop scheduling problem is given in Section II. Detailed description of the proposed job shop scheduling algorithm is presented in Section III. Section IV discusses the experimental results. Finally, we summarize the paper and present our future work in Section V.

II. JOB SHOP SCHEDULING PROBLEM

The problem studied in the paper is a deterministic and static n-job, m-machine JSSP. In this problem, n jobs are to be processed by m machines. Each job consists of a predetermined sequence of task operations, each of which needs to be processed without preemption for a given period of time on a given machine. Tasks of the same job cannot be processed concurrently and each job must visit each machine exactly once. Each operation cannot be commenced until the processing is completed, if the precedent operation is still being processed. A schedule is an assignment of operations to time slots on the machines. The makespan is the maximum completion time of the jobs. The objective of the JSSP is to find a schedule that minimizes the makespan.

Explaining the problem more specifically, let \( J = \{1, 2, ..., n\} \) denote the set of jobs, \( M = \{1, 2, ..., m\} \) represent the set of...
machines, and $O = \{0, 1, 2, \ldots, n \times m, n \times m + 1\}$ be the set of operations to be scheduled, where 0 and $n \times m + 1$ represent the dummy initial and final operations, respectively. The operations are interrelated by the precedence constraints, which force each operation $j$ to be scheduled after all predecessor operations $P_j$ are completed. Moreover, operation $j$ can only be scheduled if the required machine is idle. Furthermore, let $T_j$ and $F_j$ denote the fixed processing time and the finish time of operation $j$, respectively. Let $A(t)$ be the set of operations being processed at time $t$, and let $e_{km} = 1$ if operation $j$ is required to process on machine $m$ ($e_{km} = 0$ otherwise).

The conceptual model of the JSSP can be stated as [13]

$$\min F_{\text{new}}$$

s.t. $F_j \leq F_{j'} - T_j, \quad j = 1, 2, \ldots, n \times m + 1; \quad k \in P_j$ \hspace{1cm} (1)

$$\sum_{j \in A(t)} e_{km} \leq 1, \quad m \in M; \quad t \geq 0$$ \hspace{1cm} (2)

$$F_j \geq 0, \quad j = 1, 2, \ldots, n \times m + 1.$$ \hspace{1cm} (3)

The objective function (1) minimizes the finish time of the last operation, namely, the makespan. Constraint (2) imposes the precedence relations between operations. Constraint (3) represents that each machine can only process one operation at a time, and constraint (4) forces the finish times to be nonnegative.

III. HYBRID GENETIC ALGORITHM FOR JSSP

The GA simulates the biological processes that allow the consecutive generations in a population to adapt to their environment. The adaptation process is mainly applied through genetic inheritance from parents to children and through survival of the fittest. The GA object determines which individuals should survive, which should reproduce, and which should die. To successfully apply a GA to solve a problem one needs to determine the following [14]:

1) The representation of possible solutions, or the chromosomeal encoding;
2) The fitness function which accurately represents the value of the solution;
3) Genetic operators (selection, crossover and mutation) have to employ and the parameter values (population size, probability of applying operators, etc.) that are suitable.

A. Chromosome Representation

A proper chromosome representation has a great impact on the success of the used GA. Cheng, Gen and Tsujimura [9] gave a detailed tutorial survey on papers using different GA chromosome representations to solve classical JSSP. In this paper, an operation based representation is adopted, which uses an unpartitioned permutation with $m$ repetitions of job numbers for problems with $n$ jobs and $m$ machines. Within the representation, each job number occurs $m$ times in the chromosome. By scanning the chromosome from left to right, the $k$-th occurrence of a job number refers to the $k$-th operation in the technological sequence of this job.

For example, suppose that a chromosome is given as $[2 1 3 1 2 2 3 1 3]$ in a three jobs and three machines problem. Because each job consists of three operations, the job number occurs exactly three times in the chromosome. The fifth gene of the permutation implies the second operation of job 2 because number 2 has been repeated twice. Similarly, the sixth gene represents the third operation of job 2, and so on. The prominent advantage of operation based representation is that the permutation is always feasible. Moreover, it eliminates the deadlock schedules that are incompatible with the technological constraints and can never be finished. However, it will produce redundancy in the search space and will cause the search-space size to expand to $(n \times m)!/(n!)m^6$.

B. Chromosome Decoding

The solution of the JSSP can be represented as the operation permutation of jobs on each machine. The total number of all possible schedules (both feasible and infeasible) is $(n!)^m$ for problems with $n$ jobs and $m$ machines. Obviously, it is impossible to exhaust all the alternatives for finding the optimal solution even if the values of $n$ and $m$ are small. For example, for the Fisher-Thompson benchmark problem of ten jobs to ten machines, it has a search space with a size at about $3.96 \times 10^{65}$. Thus, it is necessary to restrict the search space and to guide the search process. The objective of the chromosome decoding procedure is to transform the chromosomes to schedules and obtain their makespans.

In general, schedules can be classified into three types: semiactive schedule, active schedule and non-delay schedule [15]. Semiactive schedules contain no excess idle time, but they can be improved by shifting some operations to the front without delaying others. Active schedules contain no idle time, and no operation can be finished earlier without delaying other operations. The set of non-delay schedules is a subset of active schedules. In a non-delay schedule, no machine is kept idle at a time when it could begin processing other operations. In order to further reduce the solution space, Zhang, Rao and Li [16] proposed a new type of schedule: full active schedule (FAS), which can be defined as a schedule with no more permissible left shifts and right shifts. Fig. 1 shows the relationships between the classes of schedules. The optimal schedule is guaranteed to be a full active schedule. Therefore, we only need to find the optimum solution in the set of full active schedules.

C. Crossover Operation

Crossover operator plays an important role in genetic
algorithm approach. It intends to inherit the properties of two parent solutions to two offspring solutions. To apply crossover operation successfully to the JSSP, we must satisfy the following criteria: completeness, feasibility, non-redundancy and characteristics preservation [4]. In this paper, we use the set-partition crossover (SPX) [17] as crossover, which can preserve characteristics properly between parents and their children. Given chromosomes, parent1 and parent2, crossover applied SPX generates the children, child1 and child2, by the following procedure. Firstly, randomly divide the set of job numbers as \{1, 2, ..., n\} into two nonempty exclusive subsets as J1 and J2. Secondly, combine together those numbers of parent1 in J1 and those numbers of parent2 in J2. The combination order is in an interweaving way, i.e. one by one from up-to-down and left-to-right. This part of procedure creates one new string. Exchange the two parents parent1 and parent2, and do the combination once again to yield another new string. Fig. 2 shows an example of the three jobs and three machines problem; chromosome of parent1 in J1 and those numbers of parent2 in J2. The crossover generates two children chromosomes, child1 \{1 2 3 3 2 1 3 2 1\} and child2 \{1 2 2 2 3 1 3 3 1\} respectively. The crossover operation successfully to the JSSP, we must satisfy the following criteria: completeness, feasibility, non-redundancy and characteristics preservation. In this paper, we use the set-partition crossover (SPX) [17] as crossover, which can preserve characteristics properly between parents and their children.

\[ \text{parent1} = \{1, 2, 3\} \quad \text{parent2} = \{3, 2, 1\} \]

\[ \text{child1} = \{1, 2, 3, 3, 2, 1, 3, 2, 1\} \quad \text{child2} = \{1, 2, 2, 2, 3, 1, 3, 3, 1\} \]

Fig. 2 Example of SPX crossover

**D. Mutation Operation**

Mutation is another important genetic operator that randomly changes a chromosome. This is done to maintain the diversity of the chromosomes and to introduce some extra variability into the population. In this paper, two types of mutation operators named forward insertion mutation (FIM) and backward insertion mutation (BIM) are used. Fig. 3 shows examples of the three jobs and three machines problem. In this work, the two mutation operators alternate randomly with equal probability. Two mutations are described as follows:

1. Forward insertion mutation selects two elements randomly and inserts the back one before the front one.
2. Backward insertion mutation selects two elements randomly and inserts the front one after the back one.

\[ \text{parent1} = \{1 2 3\} \quad \text{parent2} = \{1 2 2\} \]

\[ \text{child1} = \{1 2 3 1 3 2\} \quad \text{child2} = \{1 2 2 3 1 3 1\} \]

Fig. 3 Examples of the two mutation operators

**E. Local Search Procedure**

The use of local search techniques has been proven to be useful in solving combinatorial problems. Local search methods are applied to a neighborhood of a current solution. In the case of JSSP, a neighborhood is achieved by moving and inserting an operation in a machine sequence. In this paper, we focus particularly on the approach of Nowicki and Smutnicki [6], which is noted for proposing and implementing the most restrictive neighborhood in the literature. According to Nowicki and Smutnicki’s work, a critical path in the solution is identified first. Then the operations on the critical path are called critical operations and the maximal sequence of adjacent critical operations that are processed on the same machine can be defined as blocks. The neighborhood is defined as interchanges of the last two or the first two critical operations of the blocks if the blocks are neither the first block nor the last block. In the first block only the last two operations and symmetrically in the last block of the critical path only the first two operations are swapped. If a block contains only one operation no swap is made. The Nowicki and Smutnicki's neighborhood is illustrated in Fig. 4.

The proposed local search starts with a feasible schedule S as an input. The input schedule is set to Sbest which stands for the best found solution. Then, a single arbitrary critical path is generated and a neighborhood of schedule Sbest is constructed. Randomly select a schedule Snew from the neighborhood. If Snew is better (i.e. has a lower makespan) than Sbest, the Sbest is replaced by Snew. The procedure is repeated until a maximum number of iterations (LOC_ITER) without improving the best found solution is reached. The pseudo-code of the local search heuristic is shown in Algorithm LS.

**Algorithm LS (local search)**

1. Calculate the makespan \( C_{\text{max}}(S_{\text{best}}) \) of the current schedule S. Set iteration counter 1. count to 1
2. While (count \(<\) LOC_ITER) do
3. Randomly selected a schedule \( S_{\text{new}} \) from the neighborhood of \( S_{\text{best}} \)
4. \( \text{If} \ C_{\text{max}}(S_{\text{new}}) < C_{\text{max}}(S_{\text{best}}) \) Then
5. Update \( S_{\text{best}} \) by setting \( S_{\text{best}} = S_{\text{new}} \)
6. Set count to 1
7. Else
8. count++
9. End If
10. End While

swapping the last two operations

swapping the first two operations

Fig. 4 The Nowicki and Smutnicki’s neighborhood
F. Designing a hybrid genetic algorithm for JSSP

In contrast to a simple genetic algorithm, a new generation alternation model is introduced for the proposed hybrid GA in this paper. Every pair of randomly selected distinct mates must pass either crossover or mutation, which are deployed in parallel. The crossover is performed with a probability \( P_c \). When the mating process is carried out, crossover operator is applied to the two parents \( N \) times and \( 2N \) offspring are generated; the best individual in those offspring is selected to the next generation. Otherwise, implements the mutation operator to the two parents \( N \) times respectively and \( 2N \) offspring are generated too; the best individual is selected to the next generation. The crossover rate \( P_c \) is decreased linearly from 0.9 to 0.5 according to (5), where \( g \) represents the iterative number; \( MAX\_GEN \) is the maximum number of iterations. Such a mechanism can improve the exploration ability of GA. For example, at the beginning of the evolution period, the crossover rate is big; whereas at the end of the convergence period, the crossover rate decreases and the mutation rate becomes big; this characteristic of the new crossover rate can avoid premature convergence better.

\[
P_c = 0.9 - g \times \frac{MAX\_GEN}{MAX\_GEN} 
\]

(5)

The brief outline of the proposed algorithm can be described as follows.

Step 1) Set values of \( \text{pop}_\text{size}, N, MAX\_GEN, LOC\_ITER \).

Step 2) Generate a population \( P_0 \) with \( \text{pop}_\text{size} \) individuals randomly and evaluate the individuals with the decoding procedure; set generation counter \( g = 1 \) and the current population \( P_{old} = P_0 \).

Step 3) Repeat Step 4) – 11) until \( g > MAX\_GEN \).
Step 4) Copy the elite individual from $P_{\text{old}}$ to the new population $P_{\text{new}}$. Set the new population size $n = 1$.
Step 5) Repeat Step 6) – 9) until $n > \text{pop \_size}$.
Step 6) Select a pair of individuals $p_1, p_2$ from the $P_{\text{old}}$
Step 7) Generate a random float $\text{rand\_num} \in (0, 1)$, if $\text{rand\_num} < P_c$ go to Step 8), else go to Step 9).
Step 8) Implement crossover on $p_1$ and $p_2$ for $N$ times and generate $2N$ offspring, select the best individual in the $2N$ offspring to the next generation. Set $n = n + 1$.
Step 9) Implement mutation on $p_1$ and $p_2$ $N$ times respectively and generate $2N$ offspring, select the best individual to the next generation. Set $n = n + 1$.
Step 10) Implement local search on every individual in $P_{\text{new}}$.
Step 11) Set $P_{\text{old}} = P_{\text{new}}$

IV. COMPUTATIONAL RESULTS

To illustrate the effectiveness and performance, we use 43 instances that are taken from the ORLibrary [18] as test benchmarks to test our new proposed hybrid GA. In the 43 instances, FT06, FT10 and FT20 were designed by Fisher and Thompson in 1963 and instances LA01–LA40 that were designed by Lawerence in 1984. The algorithm was implemented in Visual C++ and the tests were run on a computer with Pentium IV 2.4G and 1GB RAM. In our experiments, population size $\text{pop \_size} = 100$, $N = 5$, LOC_ITER is the smallest integer number not less than $n/2$, and $P_c$ is decreased linearly from 0.9 to 0.5. The algorithm was terminated when after $\text{MAX \_GEN} = n^s/m$ generations of the algorithm, and each instance is randomly run 20 times. Numerical results are compared with those reported in some existing literature works using some heuristic and meta-heuristic algorithms, including HGA-param [13], LSGA [12], GRASP [7], GP+PR [19], TSAB [20], RCS [21], and SBII [22].

Table I summarizes the results of the experiments. The contents of the table include the name of each test problem (Instance), the scale of the problem (Size), the value of the best known solution for each problem (BKS), the value of the best solution found by using the proposed algorithm (our HGA) and the best results reported in other research works.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Size</th>
<th>BKS</th>
<th>Best</th>
<th>BRD (%)</th>
<th>Mean</th>
<th>MRD (%)</th>
<th>t-avg (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f06</td>
<td>6×6</td>
<td>55</td>
<td>55</td>
<td>0.00</td>
<td>55</td>
<td>0.00</td>
<td>0.62</td>
</tr>
<tr>
<td>f10</td>
<td>10×10</td>
<td>930</td>
<td>930</td>
<td>0.00</td>
<td>936.85</td>
<td>0.74</td>
<td>8.21</td>
</tr>
<tr>
<td>f20</td>
<td>20×5</td>
<td>1165</td>
<td>1165</td>
<td>0.00</td>
<td>1171.9</td>
<td>0.59</td>
<td>16.37</td>
</tr>
<tr>
<td>la01</td>
<td>10×5</td>
<td>666</td>
<td>666</td>
<td>0.00</td>
<td>666</td>
<td>0.00</td>
<td>1.90</td>
</tr>
<tr>
<td>la06</td>
<td>15×5</td>
<td>926</td>
<td>926</td>
<td>0.00</td>
<td>926</td>
<td>0.00</td>
<td>5.42</td>
</tr>
<tr>
<td>la11</td>
<td>20×5</td>
<td>1222</td>
<td>1222</td>
<td>0.00</td>
<td>1222</td>
<td>0.00</td>
<td>14.63</td>
</tr>
<tr>
<td>la16</td>
<td>10×10</td>
<td>945</td>
<td>945</td>
<td>0.00</td>
<td>947.15</td>
<td>0.23</td>
<td>7.65</td>
</tr>
<tr>
<td>la21</td>
<td>15×10</td>
<td>1046</td>
<td>1047</td>
<td>0.10</td>
<td>1057.15</td>
<td>1.07</td>
<td>24.49</td>
</tr>
<tr>
<td>la26</td>
<td>20×10</td>
<td>1218</td>
<td>1218</td>
<td>0.00</td>
<td>1218</td>
<td>0.00</td>
<td>62.48</td>
</tr>
<tr>
<td>la31</td>
<td>30×10</td>
<td>1784</td>
<td>1784</td>
<td>0.00</td>
<td>1784</td>
<td>0.00</td>
<td>202.81</td>
</tr>
<tr>
<td>la36</td>
<td>15×15</td>
<td>1268</td>
<td>1278</td>
<td>0.79</td>
<td>1286.55</td>
<td>1.46</td>
<td>56.20</td>
</tr>
</tbody>
</table>

It can be seen from Table I that the proposed algorithm is able to find the best known solution for 32 instances, i.e. in about 75% of the instances, and the deviation of the minimum found makespan from the best known solution is only on average 0.14%. The proposed algorithm yields a significant improvement in solution quality with respect to almost all other algorithms, expected for the approach proposed by Nowicki and Smutnicki that has a better performance in the $15 \times 15$ problems mainly. The superior results indicate the successful incorporation of the improved GA and LS, which facilitates the escape from local minimum points and increases the possibility of finding a better solution. Therefore, it can be concluded that the proposed hybrid GA solves the JSSP fairly efficiently.

As mentioned above, the algorithm is performed 20 times for each instance. Table II lists the best solution (Best), the relative deviation of the best solution (BRD), the mean solutions (Mean), the relative deviation of the mean solution (MRD), and the average computing time (t-avg) of some typical instances with different size. The MRD is commonly zero for small-size problem and is not more than 1.5% for most other problems.

To illustrate the simulated results more intuitively, the problem la37 that is one of the hardest problems is specially described as an example. Fig. 5 plots the representative convergence curve finding best solution. Fig. 6 shows a Gantt chart of a best solution.
V. CONCLUSION AND PERSPECTIVES

This paper presents a hybrid algorithm combining genetic algorithm with local search for the JSSP. In the algorithm a new generation alternation model of genetic algorithm for JSSP is designed and a Nowicki and Smutnicki’s neighborhood based local search algorithm is incorporated. This allows the GA to explore more solution space whereas LS does the exploitation part. The approach is tested on a set of 43 standard instances taken from the literature and compared with other approaches. The computational results show that the algorithm produced optimal or near-optimal solutions on all instances tested. Overall, the algorithm produced solutions with an average relative deviation of 0.14% to the best known solution. In our future work we aim to extend the proposed algorithm in order that it can be applied to more practical and integrated manufacturing problems such as dynamic arrivals, machine breakdown, or other factors that affect job status over time.

REFERENCES


