Journal of Computer Science 8 (5): 681-693, 2012
ISSN 1549-3636

Integration of Genetic Algorithm with Tabu Search for Job Shop Scheduling with Unordered Subsequence Exchange Crossover

Thamilselvan, R. and P. Balasubramanie
Department of Computer Science and Engineering,
Kongu Engineering College, Perundurai, Erode 638 052, Tamilnadu, India

Abstract: Problem statement: The problem of scheduling n jobs on m machines with each job having specific machine route has been researched over the decade. The Job Shop Scheduling (JSS) is one of the hardest combinatorial optimization problems. Each resource can process at most one job at a time. **Approach:** This study proposes a new approach to solve a Job Shop Scheduling problem with the help of integrating Genetic Algorithm (GA) and Tabu Search (TS). After an initial schedule is obtained the GA, the result is given as an input to TS to improve the status of the initial schedule. The objective of this study is to minimize the makespan, process time and the number of iterations. This approach achieves a better result with the help of efficient chromosome representation, powerful crossover strategies and neighborhood strategies. **Results:** This research resolves the allocation of operation to different machine and the sequence of operation based on machine sequence. Job Scheduling is the process of completing jobs over a time with allocation of shared resources. It is mainly used in manufacturing environment, in which the jobs are allocated to various machines. Jobs are the activities and a machine represents the resources. It is also used in transportation, services and grid scheduling. **Conclusion/Recommendations:** The result and performance of the proposed work is compared with the other conventional algorithm and it is also testing using standard benchmark problems.

Key words: Job Shop Scheduling (JSS), Genetic Algorithm (GA), Tabu Search (TS), Simulated Annealing (SA), Tabu List (TL), Aspiration Criteria (AC)

INTRODUCTION

Meta-heuristics is used to solve with the computationally hard optimization problems. Meta-heuristics consist of a high level algorithm that guides the search using other particular methods. Meta-heuristics are used as a standalone approach for solving hard combinatorial optimization problems. But now the standalone approach is drastically changed and attention of researchers has shifted to consider another type of high level algorithms, namely hybrid algorithms. There are at least two issues has to be considered while combining the more than one meta-heuristics: (a) how to choose the meta-heuristic methods to combine and (b) how to combine the chosen heuristic methods into new hybrid approaches. Unfortunately, there are no theoretical foundations for these issues. For the former, different classes of search algorithms can be considered for the purposes of hybridization, such as exact methods, simple heuristic methods and meta-heuristics. Moreover, meta-heuristics themselves are classified into local search based methods, population based methods and other classes of nature inspired meta-heuristics. Therefore, in principle, one could combine any methods from the same class or methods from different classes. Our hybrid approach combines Genetic Algorithms (GAs) and Tabu Search (TS) methods. Roughly, our hybrid algorithm runs the GA as the main algorithm and calls TS procedure to improve individuals of the population. The rest of the study is organized as follows.We briefly present the problem description and formulation. Followed by we have discussed about the literature review. In fourth part, GA and TS methodologies are given for job shop scheduling. Finally implementation of the HGATS to the JSSP is given with the algorithm using the proposed method and the experimental results and a discussion of the proposed method are given and a conclusion and future enhancement is also given.

Problem description and formulation: The nxm Job Shop Scheduling problem labeled by the symbol n, m, J, O, G and C_{max}. It can be described by the finite set of n jobs $J = \{j_0, j_1, j_2, j_3,\ldots..j_n, j_{n+1}\}$ (the operation 0 and n+1 has duration and represents the initial an final operations), each job consist of a chain of operations O

Corresponding Author: Thamilselvan, R., Department of Computer Science and Engineering,
Kongu Engineering College, Perundurai, Erode 638 052, Tamilnadu, India

= {o_1,o_2,o_3,….o_m}, Each operation has processing time {λ_{i1}, λ_{i2}, λ_{i3},…. λ_{im}}, finite set of m machines M = {m_1, m_2, m_3….m_m}, G is the matrix that represents the processing order of job in different machines and C_{max} is the makespan that represents the completion time of the last operation in job shop. On O define A, a binary relation representing precedence between operations. If then u has to be performed before v. A schedule is a function that for each operation u defines a start time S (u). A schedule S is feasible if it satisfy the condition in Eq. 1-4:

$$\forall u \in 0 : S(u) \geq 0 \quad (1)$$

$$\forall u, v \in 0, (u, v) \in A : S(u) + \lambda(u) \leq S(v) \quad (2)$$

$$\forall u, v \in 0, u \neq v, M(u) = M(v) : + \\ \lambda(u) \leq S(v) \text{or} S(v) + \lambda(v) \leq S(u) \quad (3)$$

$$\text{The length of a schedule S is len(S)} = \max_{v \in 0}(S(u) + \lambda(u)) \quad (4)$$

The goal is to find an optimal schedule, a feasible schedule of minimum length, min (len (S)).

An instance of the JSS problem can be represented by means of a disjunctive graph G = (O, A, E).The vertices in O represent the operations, the conjunctive arcs in A represent the given precedence between the operations and the edge in E = {(u, v)| u, v∈ 0,u ≠ v, M (u) = M(v)}represent the machine capacity constraints. Each vertex u has a weight, equal to the processing time λ (u). Let us consider the bench mark problem of the JSSP with four jobs, each has three different operations and there are three different machines. Operation sequence, machine assignment and processing time are given in Table 1.

Based on the above bench mark problem, we create a matrix *G*, in which rows represent the processing order of operation and the column represents the processing order of jobs. Also we create a matrix P, in which row *i* represents the processing time of Ji for different operations:

$$G = \begin{bmatrix} M_1 & M_2 & M_3 \\ M_3 & M_2 & M_1 \\ M_2 & M_3 & M_1 \\ M_1 & M_3 & M_2 \end{bmatrix} P = \begin{bmatrix} 2 & 3 & 4 \\ 4 & 4 & 1 \\ 2 & 2 & 3 \\ 3 & 3 & 1 \end{bmatrix}$$

The processing time of operation i on machine j am represented by O_{ij}. Let λ_{ij} be the processing time of O_{ij} in the relation. C_{ij} represents the completion of the operation O_{ij}. So that the value $C_{ij} = C_{ik} + \lambda_{ij}$ represents the completion time of O_{ij}. The main objective is to minimize of C_{max}. It can be calculated as Eq. 5:

$$C_{max} = \max_{all} 0_{ij} \in 0(C_{ij}) \quad (5)$$

The distinctive graph of the above bench mark job scheduling problem is shown in Fig. 1, in which vertices are represents, the operation. Precedence among the operation of the same job is represented by Conjunctive arc, which are doted directed lines. Precedence among the operation of different job is represented by Disjunctive arc, which undirected solid lines. Two additional vertices S and E represented the start and end of the schedule.

The gantt chart of the above bench mark job scheduling problem is shown in Fig. 2. Gantt chart is the simple graphical representation technique for job scheduling. It simply represents a graphical chart for display schedule; evaluate makespan, idle time, waiting time and machine utilization.

Literature review: Number of researchers has adopted GA and TS technique for solving the job shop scheduling problem. They include algorithms such as Simulated Annealing (SA), Genetic Algorithms (GA) (Yamada and Nakano, 1996; Gholami and Zandieh, 2009), Tabu Search (TS) (Glover, 1989; Amico and Trubian, 1993; Nowicki and Smutnicki, 1996; Thomsen, 1997; Pezzella and Merelli, 2000), ant optimisation and Genetic Local Search (GLS) (Yamada and Nakano, 1996; Zhou *et al*., 2009), Scatter Search and Path Relinking (SS PR). The majority of GA and GLS approaches appear to give poor results due to the difficulties they have with crossover operators Tabu search was first presented by (Glover, 1986) and improved in the following years. The effectiveness of the technique in the job shop problem was examined by Taillard (1994); Laarhoven *et al*. (1992); Barnes and Chambers (1995); Amico and Trubian (1993) and finally Nowicki and Smutnicki (1996). All algorithms demonstrated outstanding results comparing to simulated annealing and shifting bottleneck. Calderia *et al*. (2004) presented Tabu-Hybrid using one of the representation for the JSSP called Permutation With Repetition (PWR) in which the order of operations within the permutation is interpreted as a sequence for building a schedule solution. Yu and Liang (2001) integrate GA with neural network for JSSP. Weckman *et al*. (2008) given solution for JSSP using neural network. Eswaramurthy and Tamilarasi (2009) presented Hybridization of Ant Colony Optimization Strategies in Tabu Search for Solving Job Shop Scheduling Problem.

Table 1: Processing Time and Sequence for 4×3 problem Instance

Job	Operation number and processing sequence	Machine assigned	Processing Time
Start operation (dummy)	0	--	0
J1	O11	M1	2
	O12	M2	3
	O13	M3	4
J2	O21	M3	4
	O22	M2	4
	O23	M1	1
J3	O31	M2	2
	O32	M3	2
	O33	M1	3
J4	O41	M1	3
	O42	M3	3
	O43	M2	1
End operation (dummy)	0	--	0

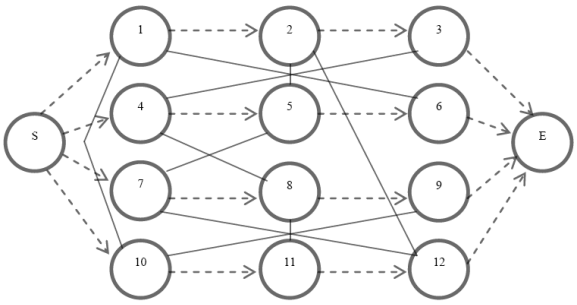

Fig. 1: Illustration of disjunctive graph

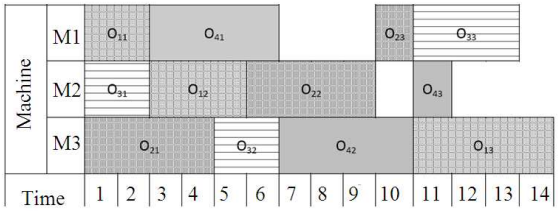

Fig. 2: A Schedule of Gantt Chart for 4X3 problem Instance

Gonzalez *et al*. (2009) presented a hybrid GA and TS system as in the case of Meeran and Morshed (2011), however (Gonzalez *et al*., 2009)"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 6instances 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. As mentioned earlier, here we are presenting in this study another aspect of the system with regard to its application to real life practical cases from real life manufacturing companies.

MATERIALS AND METHODS

Genetic algorithm: Genetic algorithms are probabilistic Meta heuristic technique, which may be used to solve optimization problems. They are based on the genetic process of chromosome. Over many generations, natural populations evolve according to the principles of natural selection, i.e., survival of the fittest, first clearly stated by Charles Darwin in The Origin of Species. It starts with the initial solution called Population and it is filled with chromosome. Each element in chromosome is called gene. Job is represented by each gene in chromosome and the job sequence in a schedule based on the position of the gene. GA uses Crossover and Mutation operation to generate a new population. By crossover operation, GA generates the neighborhood to explore new feasible solution.

A typical genetic algorithm is illustrated in Fig. 3. It first creates an initial population consisting of randomly generated solutions. After applying genetic operators, namely selection, crossover and mutation, one after the other, new offspring are generated. Then the evaluation of the fitness of each individual in the population is conducted. The fittest individuals are selected to be carried over next generation. The above steps are repeated until the termination condition is satisfied. A GA is terminated after a certain number of iterations or if a certain level of fitness value has been reached.

The construction of a genetic algorithm for the scheduling problem can be divided into four parts: The choice of representation of individual in the population; the determination of the fitness function; the design of genetic operators; the determination of probabilities controlling the genetic operators.

Algorithm: GA_Procedure:

Step 1: /*Initialization*/
Initialize 0 to MAX
Get the value for NUM

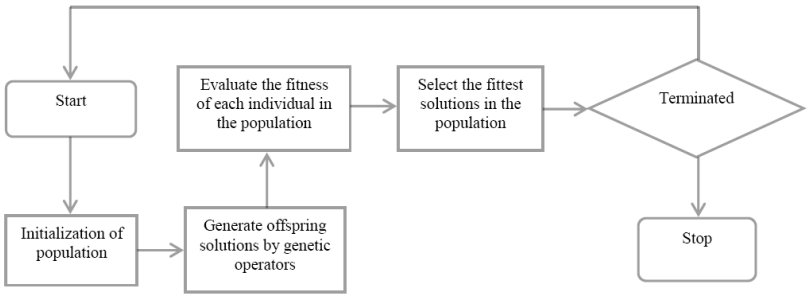

Fig. 3: A standard genetic algorithm

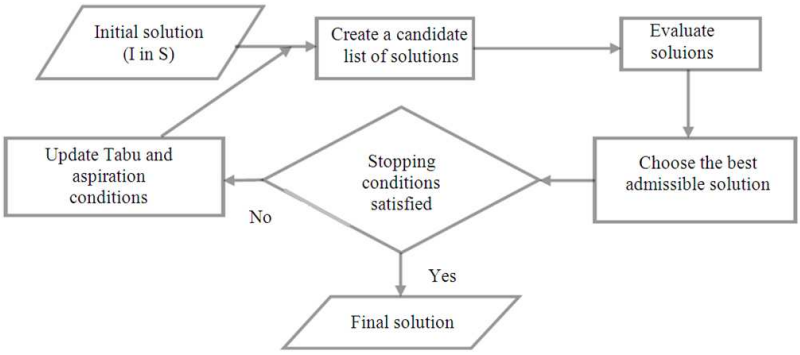

Fig. 4: A standard Tabu Search algorithm

Step 2: /*Generate initial population*/
Generate initial population using Current_Population ()
Step 3: /*Evaluate the chromosomes*/
Evaluate the fitness value of current chromosomes using Evaluate_Current ()
Step 4: Repeat
For i=0 to NUM do
For j=0 to 4 do
pop_next[j]=pop_current[j]
End for
Sorting the given set of pop_current chromosomes.
Step 5: /*Reproduction strategies*/
Apply any one Crossover strategies to get child chromosomes.
Step 6: /*Mutation strategies*/
Apply Mutation () to mutate with low probability
Step 7: /*Copy the chromosomes of pop_next to pop_current*/
For j=0 to 4 do
pop_current[j] = pop_next[j]
End for
Increment MAX by 1
Step 8: /*Termination criteria*/
If MAX<NUM then
Go to Step 4
Else
Go to Step 8
Step 9: /*Output the solution*/
Stop.

Tabu search: A typical tabu search algorithm is illustrated in Fig. 4. Tabu Search (TS) is a meta-heuristic approach used to solve combinatorial optimization problems. TS algorithm starting from initial solution and iteratively generate a new solution through its neighborhood. In TS acceptance of moving to new solution in neighborhood is deterministic. It is one of the most efficient local search algorithms for job scheduling problems. It consists of the tabu list, aspiration criteria, neighborhood structures, the move attributes and stopping rules. Tabu list the list of records that move.

Tabu List (TL) is controlled by the trial solutions in the order in which they are made. Each time a new element is added to the 'bottom' of a list, the oldest element on the list is dropped from the 'top'. Empirically, TL sizes which provide good results often grow with the size of the problem and stronger restrictions are generally coupled with smaller size.

Best sizes of TL lie in an intermediate range between these extremes. The length of the tabu list is initially assigned according to the size of the problem and it will be decreased and increased during the construction of the solution so as to achieve better exploration of the search space.

Aspiration Criteria (AC): Is another important element of TS arises when the move under consideration has been found to the associated with each entry in the TL. The simplest and most commonly used aspiration criterion consists of allowing a move, even if it is in tabu and results in a solution with an objective value better than that of the current best-known solution. Many more complicated criteria have been implemented by different researchers and successfully implemented.

Stopping Criteria (SC): The most commonly used stopping criterion in TS are either (i) a fixed number of iterations or ii) after some number of iterations without an improvement in the objective function value or (iii) when the objective function reaches a pre-specified threshold value.

Proposed algorithm: The objective of the proposed system is to minimize the make span (Cmax*)* criterion, processing time and the number of iteration while satisfying all constraints. Genetic algorithm is capable of doing a parallel search to discover the global search space. Through the parallel search mechanism GA retains useful information about what has been learned from previous generations. GA searches the solution from a population of points instead of a single point. The algorithm is computationally simple and powerful. Tabu Search (TS) works on the individual string, which are points on the solution space. TS guides (Glover, 1989; Barnes and Chambers, 1995) the iterations from one neighborhood point to another by locally improving the solution" s quality and has the ability to avoid poor local minima. Integration of GA and TS using their own strengths has a good chance of providing a reasonable solution to global combinatorial optimization problems such as JSSP. During the hybrid search process, GA starts with a set of initial solution and generates a set of new solutions. On each set of new solution, TS performs a local search to improve them. Then GA uses the improved solution of TS to continue with parallel evolution.

Hybrid Genetic Algorithm and Tabu Search (HGATS) methodology:

Proposed Hybrid Algorithm Approach:

Algorithm: HGATS_Procedure

Step 1: /*Initialization*/

Initialize 0 to MAX

Get the value for NUM

Step 2: /*Generate initial population*/

Generate initial population using Current_Population()

Step 3: /*Evaluate the chromosomes*/

Evaluate the fitness value of current chromosomes using Evaluate_Current()

Step 4: Repeat

For i=0 to NUM do

For j=0 to 4 do

pop_next [j] = pop_current [j]

End for

Sorting the given set of pop_current chromosomes.

Step 5: /*use tabu search to generate new members*/

Using tabu search algorithm, generate new members in the new poplation

Step 6: /*Reproduction strategies*/

Apply USXX Crossover strategies to get child chromosomes.

Step 7: /*Mutation strategies*/

Apply Mutation() to mutate with low probability

Step 8: /*improve status*/

Improve the status of new population by DynamicTabu() algorithm

Step 9: /*Copy the chromosomes of pop_next to pop_current*/

For j=0 to 4 do

pop_current[j] = pop_next[j]

End for

Increment MAX by 1

Step 10: /*Termination criteria*/

If MAX<NUM then

Go to Step 4

Else

Go to Step 8

Step 11: /*Output the solution*/

Stop

Procedure Current_Population is used to generate the new population and the new population is stored in pop_current variable.

Procedure: Current_population ()

Inputs: RANDOM – is a random number generated by random () function

Output: Fitness value for pop_current

Begin

Assign VALUE and RANDOM

/*Calculate fitness function*/

For i=0 to 4 do

For j=0 to 6 do

Create random value for RANDOM

RANDOM=RANDOM%2

```
            pop_current[i].bit[j] = RANDOM
        End for
    VALUE=Evaluate_Current (pop_current [i])
    /*get the value of chromosome as integer*/
    pop_current [i].fit=Calculate_Fitness(VALUE)
    /*calculate the fitness value*/
    End for
      Stop
```

Procedure Evaluate_current ():

```
        Input: pop_current[i] – for i=0 to 4
        Output: value of chromosomes as integer
         Begin
         z=pop_current.bit[0]*1 + pop_current.bit[1]*2 + pop_current.bit[2]*4 + pop_current.bit[3]*8 + pop_current.bit[4]*16
If pop_current.bit[5]==1 Then
            z=z*(-1)
         End if
         End
```

Procedure Calcuate_Fitness ():

```
        Inputs: VALUE ie value of chromosome as integer
        Output: objective function of chromosome
        Begin
          y= -(VALUE*VALUE)+5
        End
```

The crossover operator involves the swapping of genetic material (bit-values) between the two parent strings. Two parents produce two offspring. There is a chance that the chromosomes of the two parents are copied unmodified as offspring. There is a chance that the chromosomes of the two parents are randomly recombined (crossover) to form offspring. Generally the chance of crossover is between 0.6 and 1.0 (Man *et al*., 1999).

Representation: GA requires an appropriate chromosome (ie., a collection of operations) to find a solution (Cheng *et al*., 1996). All the chromosomes must be generated during the evolutionary process for feasible solution. In a traditional JSSP consist of J jobs and M machines starting that JxM operation. The chromosome [g_1, g_2, g_3, g_4, g_{JxM}] can represent a schedule of JxM operations. The chromosome could be generated based on sequence of operations.

Once the basic schedule is generated, we need GA"s crossover and mutation to generate a further schedule.
The fundamental crossover of GA operates on two parent chromosome and generates two child chromosomes. This operation needs to present the job sequence characteristics.

The representation of chromosome for JSSP is based on Cheng *et al*. (1996). The chromosome is an ordered sequence of job/operations where each gene represents a one operation. Order of the operations represented in chromosome is the order of schedule. Let us consider an example of 4X3 job shop problem. Each job shop problem has constraints for scheduling the operation to the machine with processing time is shown in Table 1. For example J1 is processed in the order M1, M2 and M3 and J2 is processed in the order M3, M2 and M1 respectively and so on. The objective of the algorithm is to complete all the operations of a particulate job with minimum possible time. Also the processing of operations on machines taking into account of the precedence and processing time of operations.

The main idea is how to represent the jobs in terms of sequence. In the relationship between the job scheduling and the chromosomes to represent the schedule. So that we can use the GA to find better job scheduling. For the above 4×3 job shop scheduling the chromosome such as [3 4 1 2 1 4 3 4 1 2 3 2] may be formed and then change the order for the better schedule. In the given chromosome the genes „1" stands for J1, „2" stands for J2 and so on. The order of the operation corresponds to the relative position of the gene. For example the first gene "3" stands for first operation of J3, seventh gene "3" stands for the second operation of J3, second gene "4" stands for first operation of J4 and so on. The above scheduling chromosome is also represented as [O_{31}, O_{41}, O_{11}, O_{21}, O_{12}, O_{42}, O_{32}, O_{43}, O_{13}, O_{22}, O_{33}, O_{23}]. O_{ij} stands for the jth operation of the job J_i. For example O31 stands for the first operation of J3.

Reproduction strategies: The crossover operator involves the swapping of genetic material (bit-values) between the two parent strings. Two parents produce two offspring. There is a chance that the chromosomes of the two parents are copied unmodified as offspring. There is a chance that the chromosomes of the two parents are randomly recombined (crossover) to form offspring. Generally the chance of crossover is between 0.6 and 1.0 (Zhang and Wu, 2008). The following sections propose the new crossover algorithms for job shop scheduling.

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.

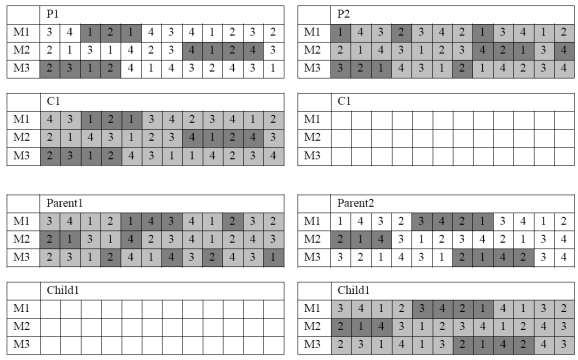

Fig. 5: Unordered Subsequence Exchange Crossover (USXX)

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.

Unordered Subsequence Exchange Crossover (USXX): We introduce one more cross over strategy named as Unordered Subsequence Exchange Crossover (USXX) that children inherit subsequences on each machine as far as possible from parents. Unordered Subsequence exchange crossover creates new children"s even the subsequence of parent1 is not in the same order in parent2. The algorithm for USXX is as follows.

Step 1: Generate two random parent individual namely P1 and P2 with a sequence of all operations
Step 2: Generate two child individual namely C1 and C2
Step 3: Select random subset of operations (genes) from P1 and copy it into C1
Step 4: Starting from the first crossover point from P1, look for elements in P2 that have been copied as in the same order
Step 5: The remaining operations of P2 that are not in the subset can be filled in C1 so as to maintain their relative ordering
Step 6: If C1 is created then goto Step 3 to generate C2 analogously

For example in Fig. 5 parent chromosome of M1 is [3, 4, 1, 2, 1, 4, 3, 4, 1, 2, 3, 2]. The selected sequence is [1, 2, 1]. It is the first operation of J1, first operation of J2 and the second operation of J1 respectively. Select the same operation from P2 even it is in different order.

In a given sample first operation of J1 is in first gene, first operation of J2 is in fourth gene and the second operation of J1 is in eighth gene respectively. Copy the remaining operation of P2 in to C2 so as to maintain their relative ordering.

The tabu length is changed during the solution construction phase to increase the exploration of the search space and this strategy called "dynamic tabu length strategy" is applied in the proposed algorithm. The proposed algorithm to find the tabu length dynamically according to the iteration number is given below. Where the inputs are the current iteration number N, m, n, δ, α, β, p and q and the output is Tabu Length (TL).

Algorithm for DynamicTabu ():

```
Start
If N < δ Then
  TL = m + n, Return TL
Else
While q < n)
  If N >= (p*δ) and N < (p*δ + q*α) Then
  TL = (m + n) + (q*β), Return TL
  Else
q = q + 1
      EndIf
    EndWhile
    EndIf
    End
```

The range, α and β are calculated as given in the Eq. 6-8 respectively. The integer parts of these variables are used for processing. p and q are the control variables used to find the position of the current iteration within the range interval:

$$\delta = TOTN / (2 * m) \quad (6)$$

$$\alpha = \delta / (m + n) \quad (7)$$

$$\beta = (\alpha + 2 * m) / (m + n) \quad (8)$$

The number of jobs n and the number of machines m are also given as inputs. The value of the Tabu Length (TL) is m + n for the first range of iterations. For the even and odd range intervals, TL value is increased and decreased respectively by the value of β with subsequent interval value of α. This strategy improves the performance of the tabu search during the construction of the solution. TOTN represents the total number of iterations. TOTT represents the maximum number of times for which the improvement is not made during the construction of the solution. The length of the tabu list is dynamically changed by using the procedure

DynamicTabu () according to the current iteration number. If the selected neighbor s_i (0<i< k) is not in the tabu or the aspiration criterion is met, the neighbor si is added to the tabu. The aspiration criterion is used to check the condition f(S) < f (S∗). f (S) is the makespan of the neighborhood solution S produced by the application of the neighbor si which is already in the tabu and f (S∗) is the current best known solution. If the neighbor cannot be added to the tabu, the tabu list is cleared and the tabu restrictions are removed. This process is repeated until a termination criterion is met. The termination criterion is either reaching the maximum iterations or no improvements of the constructed solution for the TOTT number of iterations.

RESULTS AND DISCUSSION

To measure the effectiveness of the proposed algorithm, we consider the standard JSP test instances of Fischer and Thomson (1963) instances FT06, FT10, FT20, instances from LA01 to LA40, instances SWV01-SWV20 and Yamada and Nakano (1996) instances YN1-YN4. The proposed algorithm is compared with Tabu Search (Nowicki and Smutnicki, 1996), Genetic Algorithm (Gonçalves and Beirao). The proposed algorithm is implemented using C++ programming language on windows platform with Intel Pentium E5800, 3.2 GHz and 2GB RAM. The performance of the proposed algorithm ids based on the Relative Percentage Deviation (RPD) which is computed as.

Where Algosol is the solution obtained by different existing and proposed algorithms and Optsol is the optimal or best known solution.

Here the computational results are given for well-known bench mark problems with Tabu search, Genetic Algorithm and HGATS.

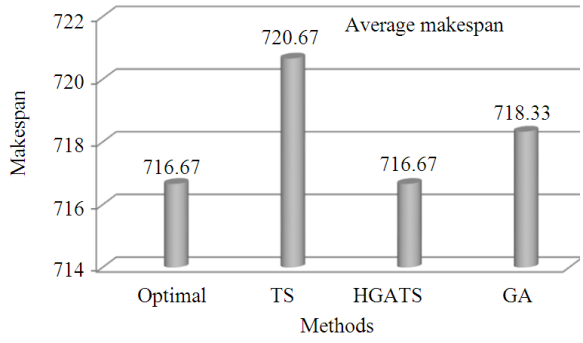

Fig. 6: Average Makespan values by Different Crossover Strategies for FT06, FT10 and FT20

Table 2 shows comparison of makespan value produced from different algorithms for problem instances FT06, FT10 and FT20 (Fisher and Thompson, 1963) Column 1 specifies the problem instances, Column 2 specifies the number of jobs, Column 3 shows the number of machines, Column 4 specify the optimal value for each problem. Column 5-7 specify results from TS, HGATS and GA respectively. It shows that HGATS with USXX strategy has succeeded in getting the optimal solutions for all the problems.

Figure 6 shows average makespan value generated by TS, HGATS and GA for different problem instances. It also shows that TS produce the worst result compare to other two algorithms and the HGATS algorithm is better than the other two algorithms. Figure 7 shows the comparison of Average Relative Error for all the three methods. It clearly shows that the Average Relative Error for HGATS is zero.

Table 3 shows comparison of makespan value produced from different algorithms for problem instances LA01-LA20 Column 1 specifies the problem instances, Column 2 specifies the number of jobs, Column 3 shows the number of machines, Column 4 specify the optimal value for each problem. Column 5, 6 and 7 specify results from TS, HGATS and GA respectively. It shows that HGATS with USXX strategy has succeeded in getting the optimal solutions for all the problems.

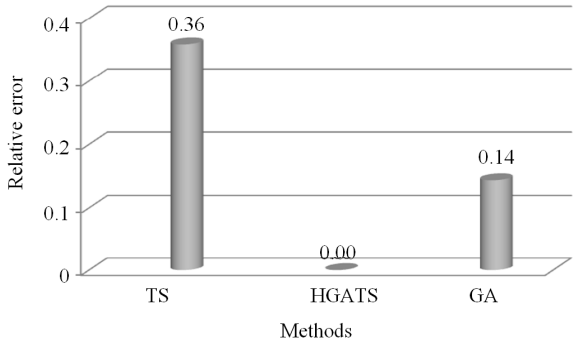

Fig. 7: Average relative error values by different crossover strategies for FT06, FT10 and FT20

Table 2: Results for instances

Problem name	Problem size: Jobs (n)	Machines (m)	Makespan time: Optimal	TS	HGATS	GA	Relative error (%): TS	HGATS	GA
FT06	6	6	55	55	55	55	0.00	0.00	0.00
FT10	10	10	930	932	930	930	0.22	0.00	0.00
FT20	20	5	1165	1175	1165	1170	0.86	0.00	0.43
Average			717	721	717	718	0.36	0.00	0.14

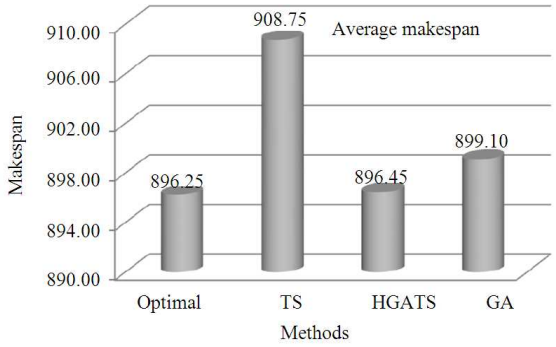

Fig. 8: Average Makespan values by different crossover strategies for LA01-LA20

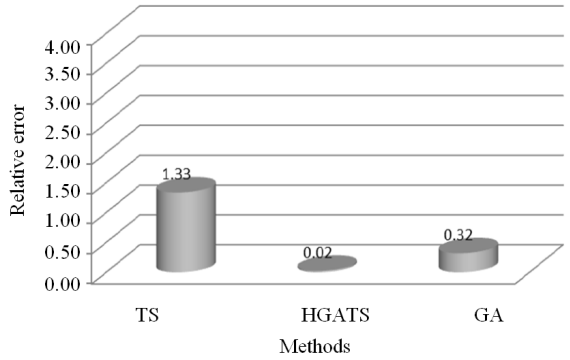

Fig. 9: Average Relative Error values by Different Crossover Strategies LA01-LA20

Figure 8 shows average makespan value generated by TS, HGATS and GA for different problem instances. It also shows that TS produce the worst result compare to other two algorithms and the HGATS algorithm is better than the other two algorithms. Figure 9 shows the comparison of Average Relative Error for all the three methods. It clearly shows that the Average Relative Error for HGATS is 0.05.

Typical runs of problem instances LA04, LA12 & LA16 are illustrated in Fig. 10-12 respectively by the GA, TS and proposed HGATS. In all cases HGATS reach the optimal solution faster than other two methods. For LA04, GA never produces the optimal solution. But GA and HGATA both are produced the optimal solution, whereas GA reached the optimal solution at 5500th iteration and HGATS reached the optimal solution at 4000th iteration. Similarly for LA12, TS reached the optimal value at 1600th iteration; GA reached the optimal value at 1500th iteration whereas HGATS reached at 1000th iteration.

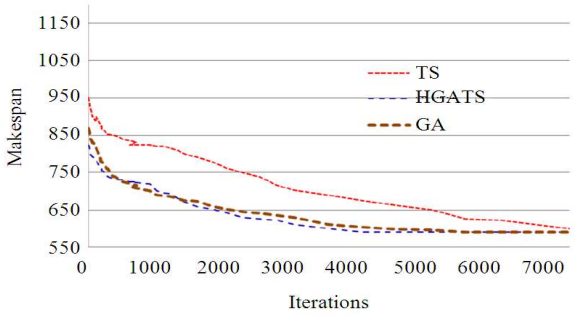

Fig. 10: The time evolutions of makespans for the LA04 (10 jobs and 5 machines)

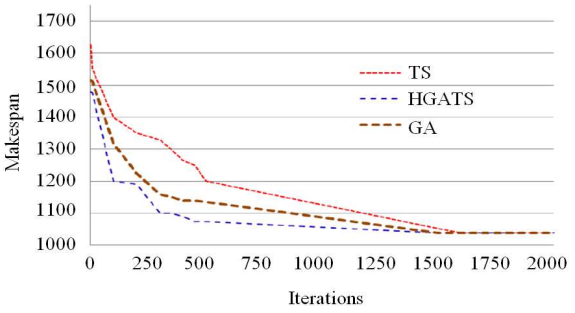

Fig. 11: The time evolutions of makespans for the LA12 (20 jobs and 5 machines)

Table 3: Results for instances Lawrence, 1984

Problem name	Problem size: Jobs (n)	Problem size: Machines (m)	Optimal	Makespan time: TS	Makespan time: HGATS	Makespan time: GA	Relative error (%): TS	Relative error (%): HGATS	Relative error (%): GA
LA01	10	5	666.00	666.00	666.00	666.00	0.00	0.00	0.00
LA02	10	5	655.00	664.00	655.00	658.00	1.37	0.00	0.46
LA03	10	5	597.00	608.00	597.00	600.00	1.84	0.00	0.50
LA04	10	5	590.00	600.00	590.00	590.00	1.69	0.00	0.00
LA05	10	5	593.00	598.00	593.00	593.00	0.84	0.00	0.00
LA06	15	5	926.00	926.00	926.00	926.00	0.00	0.00	0.00
LA07	15	5	890.00	895.00	890.00	890.00	0.56	0.00	0.00
LA08	15	5	863.00	892.00	863.00	880.00	3.36	0.00	1.97
LA09	15	5	951.00	951.00	951.00	951.00	0.00	0.00	0.00
LA10	15	5	958.00	958.00	958.00	958.00	0.00	0.00	0.00
LA11	20	5	1222.00	1222.00	1222.00	1222.00	0.00	0.00	0.00
LA12	20	5	1039.00	1039.00	103.00	1039.00	0.00	0.00	0.00
LA13	20	5	1150.00	1257.00	1150.00	1163.00	9.30	0.00	1.13
LA14	20	5	1292.00	1298.00	1292.00	1292.00	0.46	0.00	0.00
LA15	20	5	1207.00	1230.00	1207.00	1210.00	1.91	0.00	0.25
LA16	10	10	945.00	950.00	945.00	952.00	0.53	0.00	0.74
LA17	10	10	784.00	792.00	784.00	785.00	1.02	0.00	0.13
LA18	10	10	848.00	860.00	852.00	858.00	1.42	0.47	1.18
LA19	10	10	842.00	862.00	842.00	842.00	2.38	0.00	0.00
LA20	10	10	907.00	907.00	907.00	907.00	0.00	0.00	0.00
	Average		896.25	908.75	896.45	899.10	1.33	0.02	0.32

Table 4: Results for instances

Problem Name	Problem size: Jobs (n)	Problem size: Machines (m)	Optimal	Makespan time: TS	Makespan time: HGATS	Makespan time: GA	Relative error (%): TS	Relative error (%): HGATS	Relative error (%): GA
ABZ5	10	10	1234.00	1260.0	1234.0	1257.0	2.11	0.00	1.86
ABZ6	10	10	943.00	960.0	943.0	943.0	1.80	0.00	0.00
ABZ7	20	15	656.00	700.0	656.0	662.0	6.71	0.00	0.91
ABZ8	20	15	665.00	670.0	665.0	665.0	0.75	0.00	0.00
ABZ9	20	15	679.00	725.0	679.0	683.0	6.77	0.00	0.59
Average			835.40	863.0	835.4	842.0	3.63	0.00	0.67

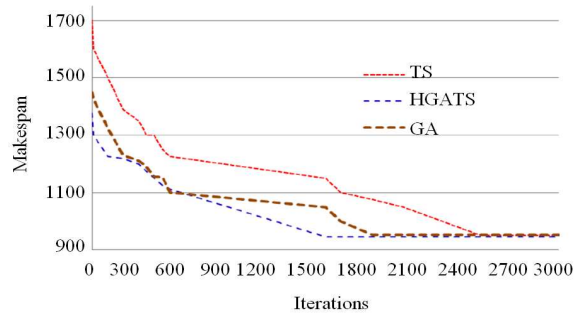

Fig. 12: The time evolutions of makespans for the LA16 (10 jobs and 10 machines)

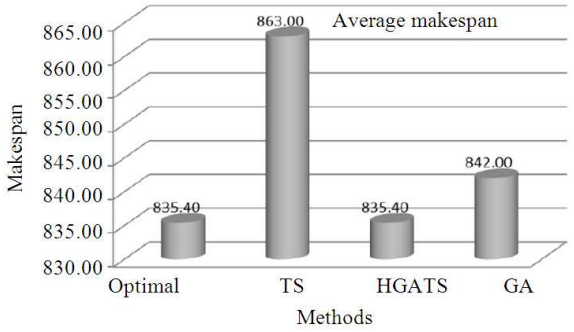

Fig. 13: Average Makespan values by different crossover strategies for ABZ5-ABZ9

Table 5: Results for instances

Problem Name	Problem size: Jobs (n)	Problem size: Machines (m)	Makespan time: Optimal	Makespan time: TS	Makespan time: HGATS	Makespan time: GA	Relative error (%): TS	Relative error (%): HGATS	Relative error (%): GA
ORB1	10	10	1059.00	1093.00	1059.00	1059.00	3.21	0.00	0.00
ORB2	10	10	888.00	903.00	888.00	888.00	1.69	0.00	0.00
ORB3	10	10	1005.00	1025.00	1005.00	1013.00	1.99	0.00	0.80
ORB4	10	10	1005.00	1012.00	1005.00	1005.00	0.70	0.00	0.00
ORB5	10	10	887.00	892.00	887.00	887.00	0.56	0.00	0.00
ORB6	10	10	1010.00	1037.00	1022.00	1025.00	2.67	1.19	1.49
ORB7	10	10	397.00	421.00	397.00	415.00	6.05	0.00	4.53
ORB8	10	10	899.00	95.00	899.00	899.00	6.23	0.00	0.00
ORB9	10	10	934.00	942.00	934.00	934.00	0.86	0.00	0.00
ORB10	10	10	944.00	952.00	944.00	962.00	0.85	0.00	1.91
	Average		902.80	923.20	904.00	908.70	2.48	0.12	0.87

Table 4-7 shows comparison of makespan value produced from different algorithms for problem instances Yamada and Nakano (1996) respectively. In all Table Column 1 specifies the problem instances, Column 2 specifies the number of jobs, Column 3 shows the number of machines, Column 4 specify the optimal value for each problem. Column 5-7 specify results from TS, HGATS and GA respectively. It values in the table shows that HGATS with USXX strategy has succeeded in getting the optimal solutions for all the problems.

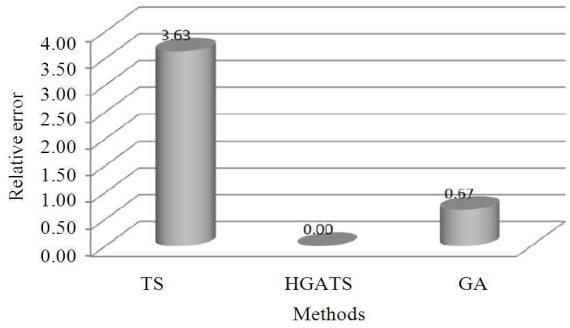

Fig. 14: Average relative error values by different crossover strategies for ABZ5-ABZ9

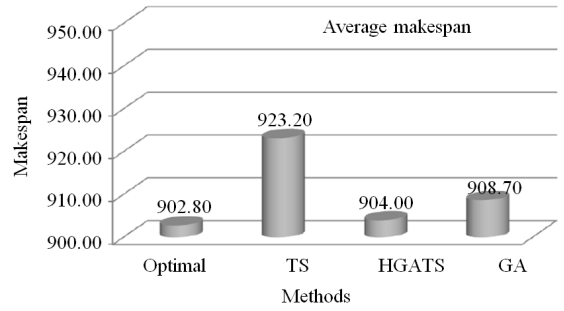

Fig. 15: Average Makespan values by different crossover strategies for ORB01-ORB10

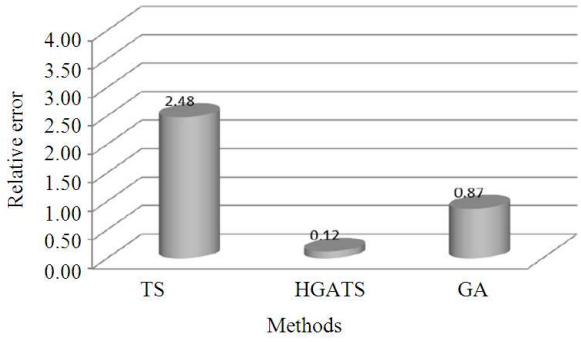

Fig. 16: Average relative error values by different crossover strategies for ORB01-ORB10

Figure 13-19 shows average makespan value generated by TS, HGATS and GA for different problem instances of Yamada and Nakano (1996) respectively. It also shows that TS produce the worst result compare to other two algorithms and the HGATS algorithm is better than the other two algorithms. Figure 14-22 shows the comparison of Average Relative Error for all the three methods. It clearly shows that the Average Relative Error for HGATS is lower than the other algorithms.

Table 6: Results for instances by Storer *et al*. 1992

	Problem size		Optimal		Makespan T time			Relative error (%)		
Problem name	Jobs (n)	Machines (m)	UB	LB	TS	HGATS	GA	TS	HGATS	GA
SWV11	50	10	2991.0	2983	3504.0	3012.00	3200.0	17.15	0.70	6.99
SWV12	50	10	3003.0	2972	3442.0	3120.00	3250.0	14.62	3.90	8.23
SWV13	50	10	3104.0		3876.0	3250.00	3754.0	24.87	4.70	20.94
SWV14	50	10	2968.0		4006.0	3212.00	3487.0	34.97	8.22	17.49
SWV15	50	10	2904.0	2885	4357.0	3589.00	4235.0	50.03	23.59	45.83
SWV16	50	10	2924.0		3986.0	3326.00	3547.0	36.32	13.75	21.31
SWV17	50	10	2794.0		3459.0	3005.00	3269.0	23.80	7.55	17.00
SWV18	50	10	2852.0		3295.0	2950.00	31156.0	15.53	3.44	992.43
SWV19	50	10	2843.0		3293.0	2934.00	3169.0	15.83	3.20	11.47
SWV20	50	10	2823.0		3329.0	2978.00	3231.0	17.92	5.49	14.45
Average			2244.2	1957.1	2661.8	2375.45	3935.7	15.78	5.21	60.14

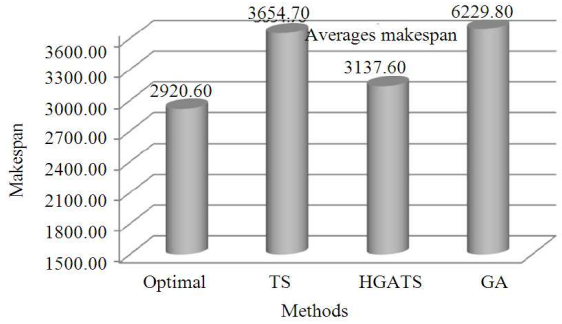

Fig. 17: Average makespan values by different crossover strategies for SWV11- SWV20

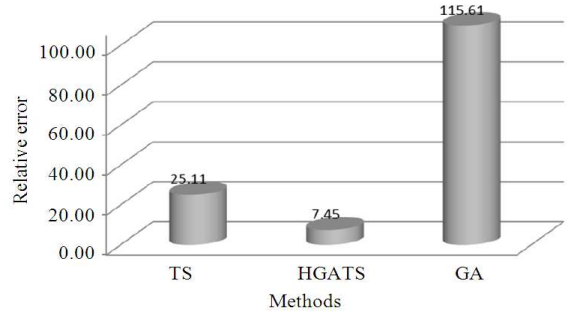

Fig. 18: Average relative error values by different crossover strategies for SWV11-SWV20

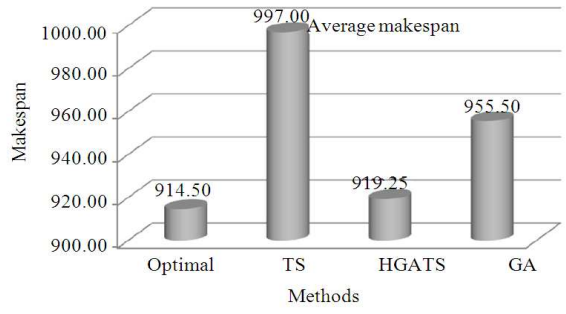

Fig. 19: Average Makespan values by different crossover strategies for YN01-YN04

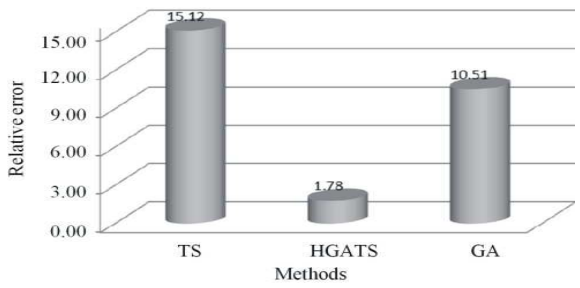

Fig. 20: Average relative error values by different crossover strategies for YN01-YN04

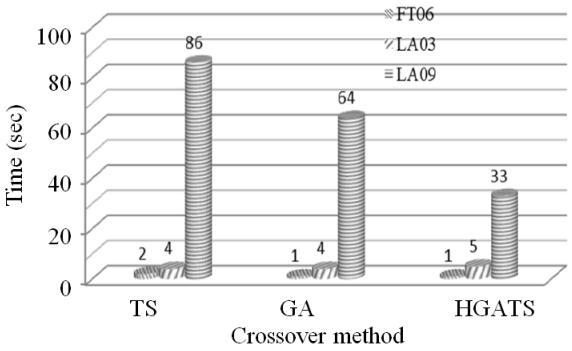

Fig. 21: Processing time for LT06, LA03 and LA09

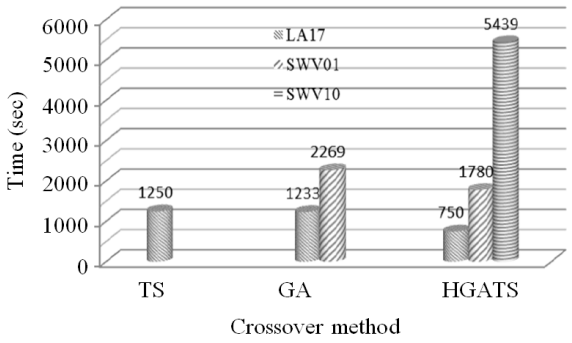

Fig. 22: Processing time for LA17, SWV01 and SWV10

Table 7: Results for instances by Yamada and Nakano (1996)

	Problem Size		Optimal		Makespan time					
Problem name	Jobs (n)	Machines (m)	UB	LB	TS	HGATS	GA	TS	HGATS	GA
YN01	20	20	888.0	826.0	895.0	888.00	890.0	7.77	0.00	7.21
YN02	20	20	909.0	861.0	925.0	909.00	910.0	7.04	0.00	5.39
YN03	20	20	893.0	827.0	1056.0	893.00	924.0	25.64	0.00	10.86
YN04	20	20	968.0	918.0	1112.0	987.00	1098.0	20.04	7.13	18.60
Average			914.5	858.0	997.0	907.50	955.5	919.25	1.78	10.51

Table 8: Comparison of CPU Time and Number of Iterations to Reach Optimal makespan using TS, GA and HGATS

	Problem size			TS			GA			HGATS		
Problem Name	Jobs (n)	Machines (m)	Optimal	CPU Time	Iterations	Makespan	CPU time	Iterations	Makespan	CPU Time	Iterations	Makespan
FT06	6	6	55	2	27	55	1	12	55	1	8	55
FT10	10	10	930	3	78	932	2	56	930	2	12	930
LA01	10	5	666	32	576	666	25	552	666	10	426	666
LA03	10	5	597	4	5952	608	4	4856	600	5	3823	597
LA04	10	5	590	12	6784	600	12	5435	590	7	4023	590
LA06	15	5	926	62	756	926	60	608	926	19	528	926
LA07	15	5	890	83	873	895	75	756	890	22	784	890
LA09	15	5	951	86	349	951	64	256	960	33	145	951
LA10	15	5	958	84	489	958	65	178	958	31	78	958
LA12	20	5	1039	206	1568	1039	150	1347	1039	62	958	1039
LA14	20	5	1292	228	256	1298	124	156	1292	67	98	1292
LA16	10	10	945	1655	2879	950	1576	1736	952	850	1238	945
LA17	10	10	784	1250	1583	792	1233	1375	785	732	967	785
LA20	10	10	907	1088	12846	907	1154	10237	907	836	8493	907
ABZ5	10	10	1234	1453	13287	1260	1322	12889	1257	1002	9457	1235
ABZ6	10	10	943	1255	10832	960	1010	9349	943	950	7584	943
ABZ7	20	15	656	1678	12359	700	NG	9484	662	1107	7345	656
ABZ8	20	15	665	NG**	NF*	670	1756	NF*	665	1298	5346	665
SWV01	20	10	1407	NG**	NF*	1430	2269	12985	1430	1657	7584	1420
SWV03	20	10	1398	NG**	NF*	1445	2567	8734	1420	1970	6483	1425
SWV07	20	15	1620	NG**	NF*	1650	NG**	NF*	1645	4378	8679	1625
SWV10	20	15	1767	NG**	NF*	1871	NG**	NF*	1855	5289	9363	1800

*: The solution could not be found; **: This information is not given

Table 8 shows the comparisons of CPU time and number of iteration to reach optimal makespan among TS, GA and HGATS for the problem instances. Column 1 provides problem instances to be used for testing whereas number of jobs and number of machines are specified in column 2 and 3 respectively. In column 4, optimal makespan value for each problem is given. Time required to reach optimal value for TS, GA and HGATS are specified in column 5-11 respectively and corresponding number of iterations are given in column 6-12 respectively. Among three methods specified in Table 3, HGATS performs well. For all problems, values of makespan are reached in HGATS with lesser time compare to TS and GA methods.

CONCLUSION

Even though many integration techniques developed for solving JSSP. Integration of TS with GA produces a better result compare to other methods. 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 proposed model has been used on different types of real-life practical problems. The system described here is able to find the optimal solutions or at least near optimal solutions for all well-known bench mark problems. In almost all cases the proposed system performed better. On all the job shop cases on which this framework has been tested improved results have been achieved. In future this algorithm may be applied with the real time application to optimize the scheduling in production. Moreover, when this system was tested on 52 benchmark problems that exist in the literature it found optimum solutions for 39 of these problems and achieved an average ARE of 1.56%.

REFERENCES

Amico, D and M.M. Trubian, 1993. Applying tabu search to the job-shop scheduling problem. Annals Oper. Res., 41: 231-252. DOI: 10.1007/BF02023076

Barnes, J.W. and J.B. Chambers, 1995. Solving the job shop scheduling problem using tabu search. IIE Transactions, 27, 257-263.

Calderia, J.P., F. Melicio and A. Rosa, 2004. Using a hybrid evolutionary-taboo algorithm to solve job shop problem. Proceedings of the ACM Symposium Applied Computing, Mar. 14-17, ACM, Nicosia, Cyprus, pp: 1446-1451. DOI: 10.1145/967900.968189

Cheng, R., M. Gen and Y. Tsujimura, 1996. A tutorial survey of job-shop scheduling problems using genetic algorithms—I. representation. Comput. Industrial Eng., 30: 983-997. DOI: 10.1016/0360-8352(96)00047-2

Chiu, H.P., K.L. Hsieh, Y.T. Tang and C.Y. Wang, 2007. A tabu genetic algorithm with search area adaptation for the job-shop scheduling problem. Proceedings of the 6th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases, (AIKED'07), ACM, Greece, pp: 16-19.

Eswaramurthy, V.P and A. Tamilarasi, 2009. Hybridization of ant colony optimization strategies in tabu search for solving job shop scheduling problems. Int. J. Inform. Manage. Sci., 20: 173-189.

Fischer, H. and G.L. Thompson 1963. Probabilistic Learning Combinations of Local Job-Shop Rules. In: Industrial Scheduling, Muth J.F. and G.L. Thompson, (Eds.). Prentice-Hall, Englewood Cliffs, NJ., pp: 225-251.

Gholami, M and M. Zandieh, 2009. Integrating simulation and genetic algorithm to schedule a dynamic flexible job shop. J. Intell. Manufact., 20: 481-498. DOI: 10.1007/s10845-008-0150-0

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Comp. Operat. Res., 13: 533-549. DOI: 10.1016/0305-0548(86)90048-1

Glover, F., 1989. Tabu Search-Part I. INFORMS J. Comput., 1: 190-206. DOI: 10.1287/ijoc.1.3.190

Gonzalez, M.A., C.R. Vela and R. Varela, 2009. Genetic algorithm combined with tabu search for the job shop scheduling problem with setup times. Proceedings of the 3rd International Work-Conference on The Interplay Between Natural and Artificial Computation: Part I: Methods and Models in Artificial and Natural Computation, (IWINAC '09), Springer-Verlag Berlin, Heidelberg, pp: 265-274.

Laarhoven, V.P.J.M., E.H.L. Aarts and J.K. Lenstra, 1992. Job shop scheduling by simulated annealing. Operat. Res., 40: 113-125. DOI: 10.1287/opre.40.1.113

Man, K.F., K.S. Tan and S. Kwong, 1999. Genetic Algorithms: Concepts and Designs. 2nd Edn., Springer, London, ISBN: 1852330724, pp: 344.

Meeran, S. and M.S. Morshed, 2011. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: A case study. J. Intell. Manuf. DOI: 10.1007/s10845-011-0520-x

Nowicki, E and C. Smutnicki, 1996. A fast taboo search algorithm for the job-shop problem. Manage. Sci., 42: 797-813. DOI: 10.1287/mnsc.42.6.797

Pezzella, F and 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. DOI: 10.1016/S0377-2217(99)00158-7

Taillard, E.D., 1994. Parallel taboo search techniques for the job shop scheduling problem. ORSA J. Comput., 6: 108-117.

Thomsen, S., 1997. Meta-heuristics combined with branch and bound. Technical Report. Copenhagen Business School, Copenhagen, Denmark

Weckman, G.R., C.V. Ganduri and D.A. Koonce, 2008. A neural network job-shop scheduler. J. Intel. Manufac., 19: 191-201. DOI: 10.1007/s10845-008-0073-9

Yamada, T. and R. Nakano, 1996. Scheduling by genetic local search with multi-step crossover. Proceeding of the 4th International Conference on Parallel Problem Solving from Nature, Sept. 22-26, Berlin, Germany, pp: 960-969.

Yu, H. and W. Liang, 2001. Neural network and genetic algorithm-based hybrid approach to expanded job shop scheduling. Comput. Ind. Eng., 39: 337-356. DOI: 10.1016/S0360-8352(01)00010-9

Zhang, R. and C. Wu, 2008. A hybrid approach to large-scale job shop scheduling. Applied Intell., 32: 47-59. DOI: 10.1007/s10489-008-0134-y

Zhou, R., A.Y.C. Nee and H.P. Lee, 2009. Performance of an ant colony job shop scheduling problems. Int. J. Produc. Res., 47: 2903-2920. DOI: 10.1080/00207540701644219