A MULTI-POPULATION BASED FROG-MEMETIC ALGORITHM FOR JOB SHOP SCHEDULING PROBLEM

Somayeh Kalantari[1] and Mohammad SanieeAbadeh[2]

[1]Department of Electrical, Computer, and Biomedical Engineering, Islamic Azad University, Qazvin Branch, Qazvin, Iran
sk_kalantari@yahoo.com

[2]Department of Electrical and Computer Engineering, Tarbiat Modares University, Tehran, Iran
saniee@modares.ac.ir

ABSTRACT

The Job Shop Scheduling Problem (JSSP) is a well known practical planning problem in the manufacturing sector. We have considered the JSSP with an objective of minimizing makespan. In this paper, we develop a three-stage hybrid approach called JSFMA to solve the JSSP. In JSFMA, considering a method similar to Shuffled Frog Leaping algorithm we divide the population in several sub populations and then solve the problem using a Memetic algorithm. The proposed approach have been compared with other algorithms for the Job Shop Scheduling and evaluated with satisfactory results on a set of the JSSP instances derived from classical Job Shop Scheduling benchmarks. We have solved 20 benchmark problems from Lawrence's datasets and compared the results obtained with the results of the algorithms established in the literature. The experimental results show that JSFMA could gain the best known makespan in 17 out of 20 problems.

KEYWORDS

Memetic algorithm, Job Shop Scheduling problem, Shuffled Frog Leaping algorithm, Multi-population

1. INTRODUCTION

Scheduling is one of the most important issues in the planning and operation of the manufacturing systems [1]. The Job Shop Scheduling problem consists of a set of jobs, job= {j_1, j_2… j_n}, and a set of machines, machine= {m_1, m_2… m_n}. In the general JSSP, each job comprises a set of tasks which must each be done on a different machine for different specified processing times, in a given job-dependent order. The standard Job Shop Scheduling problem makes the following constraints and assumptions [2]:

- 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.

Table1 shows a standard 6×6 benchmark problem (j=6 and m=6) from [3]. In this example, job 1 must go to machine 3 for 1 unit of time, then to machine 1 for 3 units of time, and so on.

Table 1.The 6×6 benchmark problem

(Machine,Time)	**(m,t)**	**(m,t)**	**(m,t)**	**(m,t)**	**(m,t)**	**(m,t)**
Job1	3,1	1,3	2,6	4,7	6,3	5,6
Job2	2,8	3,5	5,10	6,10	1,10	4,4
Job3	3,5	4,4	6,8	1,9	2,1	5,7
Job4	2,5	1,5	3,5	4,3	5,8	6,9
Job5	3,9	2,3	5,5	6,4	1,3	4,1
Job6	2,3	4,3	6,9	1,10	5,4	3,1

In this paper, the problem is to minimize the total elapsed time between the beginning of the first task and the completion of the last task, the makespan. The other measures of schedule quality exist, but shortest makespan is the simplest and most widely used criterion. For the above problem the minimum makespan is known to be 55. The schedule is shown in Figure 1 [4].

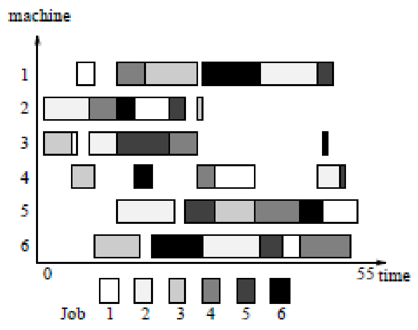


Figure 1.An optimal schedule for 6×6 JSSP benchmark

Due to the practical significance of the JSSP, it has drawn the attention of researchers for the last decades. Bruker and Schile were the first to address this problem in 1990 [5]. They developed a polynomial graphical algorithm for a two-job problem. Haung and Yin used an improved shifting bottleneck procedure for the JSSP [6]. Chen and Luh used a new alternative method to Lagrangian relaxation approach [7]. A taboo search algorithm for the JSSP was applied by Nowicki and Smutnicki [8]. Yang et al. used a clonal selection based Memetic algorithm for the JSSP [9].

The remainder of the paper is organized as follows. Section 2 describes the Memetic algorithm. The proposed approach is explained in section 3. Section 4 reports experimental results. Concluding remarks are given in section 5.

2. MEMETIC ALGORITHM

The term 'Memetic Algorithms' [10] (MAs) was introduced in the late 80s to denote a family of meta-heuristics that have as central theme the hybridization of different algorithmic approaches for a given problem .The adjective 'memetic' comes from the term 'meme', coined by R. Dawkins [11] to denote an analogous to the gene in the context of cultural evolutions. It was first proposed as a means of conveying the message that, although inspiring for many, biological evolution should not constrain the imagination to develop population-based method.

Other forms of evolution may be faster, with cultural evolution being one of those less-restrictive examples. MAs exploit problem-knowledge by incorporating pre-existing heuristics, preprocessing data reduction rules, approximation and fixed-parameter tractable algorithms, local search techniques, specialized recombination operators, truncated exact methods, etc. Also, an important factor is the use of adequate representations of the problem being tackled. This results in highly efficient optimization tools. MAs constitute an extremely powerful tool for tackling combinatorial optimization problems. Indeed, MAs are state-of-the-art approaches for many such problems. Traditional NP Optimization problems constitute one of the most typical battlefields of MAs, and a remarkable history of successes has been reported with respect to the application of MAs to such problems. Combinatorial optimization problems (both single-objective and multi-objective [12] [13] [14]) arising in scheduling, manufacturing, telecommunications, and bioinformatics among other fields have been also satisfactorily tackled with MAs [15].

MAs have been applied in a number of different areas and problem domains. It is now well established that it is hard for a 'pure' Genetic Algorithm to 'fine tune' the search in complex spaces. Researchers and practitioners have shown that a combination of global and local search is almost always beneficial [16]. So MAs which are population-based Meta heuristic search approaches have been receiving increasing attention in the recent years. Generally, MA may be regarded as a marriage between a population-based global search and local improvement procedures. It has shown to be successful for solving scheduling problems [17]. The basic steps of a canonical MA for general nonlinear optimization based on the GA can be outlined in Figure 2.

```
Procedure: Canonical-MA
Begin
Initialize: Generate an initial GA population.
While (stopping conditions are not satisfied)
        Evaluate all individuals in the population
        For each individual in the population
                Proceed with local improvement and replace the genotype and/or phenotype in the
                population with the improved solution depending on Lamarckian or Baldwinian
                learning.
        End For
        Apply standard GA operators to create a new population; i.e., crossover, mutation and
        selection.
End While
End
```

Figure2. The canonical MA pseudo-code

3. THE PROPOSED APPROACH (JSFMA)

JSFMA is a three-stage approach. Shown in Figure 3, the flowchart of the JSFMA and the components of it are described in the following.

3.1.Components of the first stage (Primary organization)

3.1.1. Parameter setting

Some parameters used in the JSFMA are chosen experimentally to get a satisfactory solution in an acceptable time span. Through experimentation, parameters values were chosen as follow:

- Number of iterations: 150
- Maximum number of iterations per temperature: 10
- Crossover probability: 0.8
- Mutation probability: 0.01

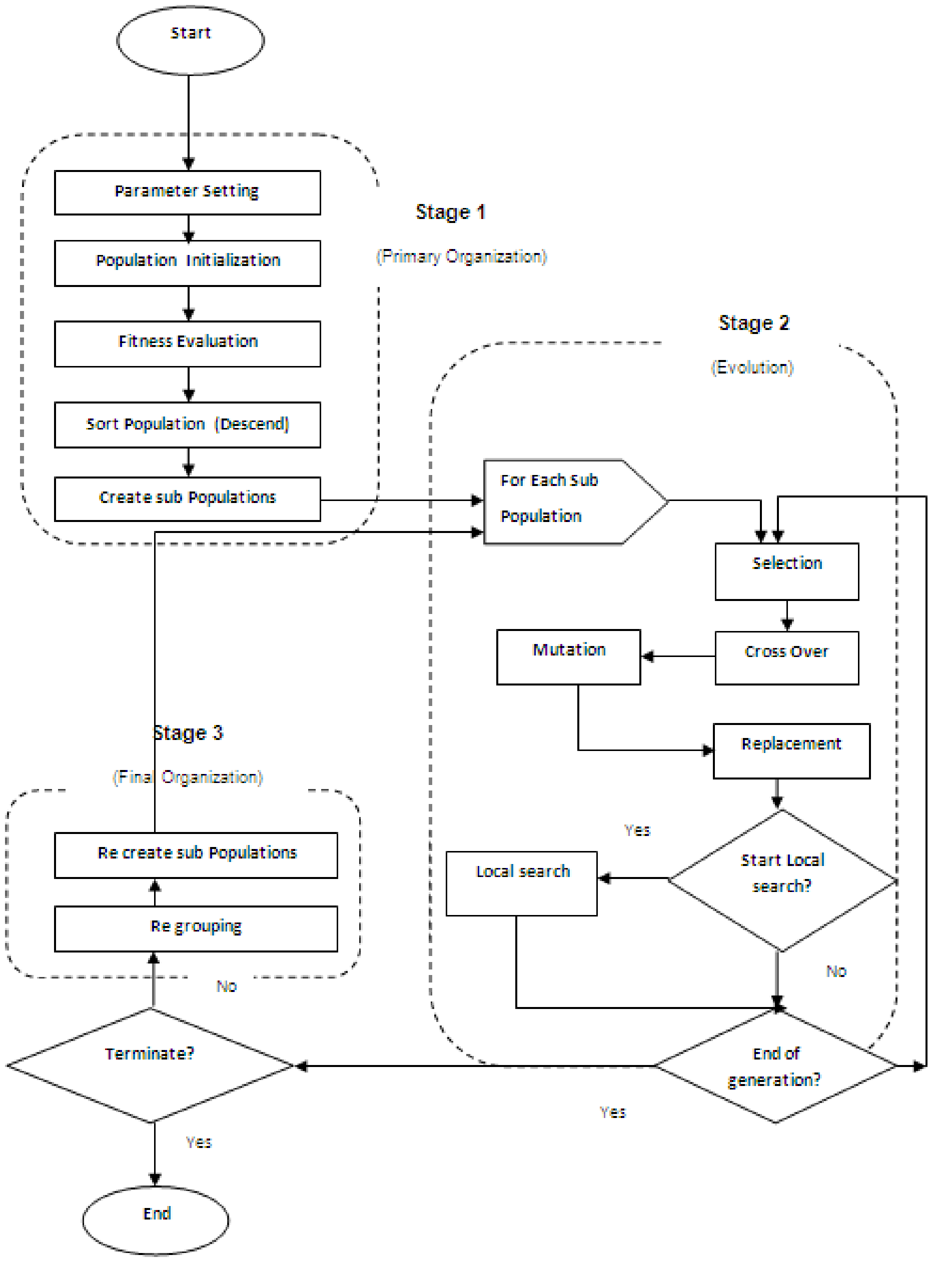


Figure 3.The flowchart of the proposed approach

Besides, each algorithm needs a large enough space to search an optimal solution. With regard to the point that we aimed to test the proposed approach on the 20 different instances with various sizes, considering a suitable search space could play an important role to find the optimal solutions in an acceptable time. The greater the sub population size, the more computation time. So analyzing the instances, we have considered the size of the entire population, the size and the number of sub populations as follow to reduce the computation time (without decreasing the quality).

- Size of the entire population: 204
- Size of each sub population: 34
- Number of sub populations: 6

3.1.2. Population initialization

In this approach, using a random way the whole population is created. A population which will be partitioned into several subsets consists of a set of solutions, schedules.

3.1.3. Fitness evaluation

The task of optimization is determining the values of a set of parameters so that some measure of optimality is satisfied, subject to certain constraints. This task is of great importance to many professions [18]. The objectives usually considered in the JSSP are the minimization of makespan, the minimization of tardiness, and the minimization of mean flow time, etc [19]. In this paper, we have considered the JSSP with an objective of minimizing makespan, a typical performance indicator for the JSSP. Makespan is defined as the total time between the starting of the first operation and the ending of the last operation in all jobs.

3.1.4. Sort population and Create Sub populations

In this part the solutions are sorted in a descending order according to their fitness. Then the entire population is divided into m sub populations, each containing n solutions. Finally, a process similar to Shuffled Frog Leaping algorithm (SFL) [20] is applied. That is, the first solution goes to the first sub population, the second solution goes to the second sub population, solution m goes to the mth sub population, and solution m+1 goes back to the first sub population, etc. These sub populations are of equal size and their boundary is closed. Of course, they can cooperate with each other in the third stage.

3.2. Components of the second stage (Evolution)

3.2.1. Selection

The selection phase is in charge to choose the better individuals for the crossover. In this paper the Roulette wheel selection method [21] is adopted. In this method, the first step is to calculate the cumulative fitness of the whole population through the sum of the fitness of all individuals. After that, the probability of selection is calculated for each individual as shown Eq.1. Then, an array is built containing cumulative probabilities of the individuals. So, n random numbers are generated in the range 0 to Pf_i and for each random number an array element which can have a higher value is searched for. Therefore, individuals are selected according to their probabilities of selection [22].

$$P\ sel_i = {}^{f_i}/_{Pf_i} \qquad (1)$$

3.2.2. Crossover

Crossover can be regarded as the backbone of genetic search. It intends to inherit nearly half of the information of two parent solutions to one or more offspring solutions. Provided that the parents keep different aspects of high quality solutions, crossover induces a good chance to find still better offspring [23].

3.2.3. Mutation

For every string that advances to the mutation component, the Inversion Mutation (IVM) [24] is used. It randomly selects a substring, removes it from the string and inserts it in a randomly selected position. However, the substring is inserted in reversed order. Consider the string (1 2 3 4 5 6 7 8) and suppose that the substring (3 4 5) is chosen. Now this substring is inserted in reversed order immediately after position 7 as shown in Figure 4 and gives (1 2 6 7 5 4 3 8) [25].

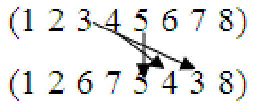


Figure 4.Inversion Mutation (IVM)

3.2.4. Replacement

The population of crossover and mutation components, new population, and the old population are sorted separately in an ascending order according to their fitness. Then using an elitist method, the first half of the new population, the best of this population, are combined with the first half of the old population, the best of the old population, to form the final population of the current generation.

3.2.5. Local search through Simulated Annealing

The local search complements to the genetic part to keep the balance of the exploitation and exploration is provided by Simulated Annealing. Simulated Annealing (SA) is motivated by an analogy to annealing in solids. The idea of SA comes from a paper published by Metropolis et al. in 1953 [26]. The algorithm in that paper simulated the cooling of material in a heat bath. In 1982, Kirkpatrick et al. [27] took the idea of the Metropolis algorithm and applied it to optimization problems. The idea is to use the Simulated Annealing to search for feasible solutions and converge to an optimal solution [28]. The SA optimization algorithm uses a similar concept. The objective function is considered as a measure of the energy of the system and this is maintained fixed for a certain number of iterations (a temperature cycle). In each of the iterations, the parameters are changed to a nearby location in parameter space and the new objective function value is calculated. If it decreases, the new state will be accepted. If it increases, the new state will be accepted with a probability that follows a Boltzmann distribution (the higher temperature means the higher probability of accepting the new state). After a fixed number of iterations, the stopping criterion is checked. If it does not come time to stop, the system's temperature will be reduced and the algorithm will continue. Simulated Annealing is a stochastic algorithm that will guarantee to converge, if it runs for an infinite number of iterations. It is one of the most robust global optimization algorithms, although it is also one of the slowest [29].

Figure 5 shows the Simulated Annealing algorithm [30]. In the Figure, the *VALUE* function corresponds to the total energy of the atoms in the material, and *T* corresponds to the temperature. The *schedule* determines the rate at which the temperature is lowered. Individual moves in the state space correspond to random fluctuations due to thermal noise. One can prove that if the temperature is lowered sufficiently slowly, the material will attain a lowest-energy (perfectly ordered) configuration. This corresponds to the statement that if *schedule* lowers *T* slowly enough, the algorithm will find a global optimum.

```
Function SIMULATED-ANNEALING (Problem, Schedule)   returns a solution state
Inputs:  Problem, a problem, Schedule, a mapping from time to temperature
Local Variables:  Current, a node
                  Next, a node
                  T, a "temperature" controlling the probability of downward steps
Current= MAKE-NODE (INITIAL-STATE[Problem])
For   t=1 to ∞ do
    T= Schedule(t)
    If T=0 then return Current
    Next= a randomly selected successor of Current
    ΔE=VALUE[Next]-VALUE[Current]
    If ΔE>0 then Current=Next
    Else Current=Next only with probability exp(-ΔE/T)
```

Figure 5.The Simulated Annealing algorithm

The local search for all chromosomes or in all generations may cost much computation time. So Ishibuchi et al. [31] proposed the following strategies:

- To apply local search to a sub set of the population selected based upon a given probability P and on the fitness of the solutions according to preset criteria.
- To apply the local search procedure not after each generation but every T>1 generations.

It has been observed that applying the local search for a limited number of iterations enables better results in the long run as reported in [32] for the maintenance scheduling problem. In the case of which solutions should be applied to each group of operators, an approach that has been used in the literature is to improve by the local search only a number of the best solutions in the population [33]. In [34] authors implemented a first version of Memetic algorithm for the Flow Shop Scheduling problem. They proposed not to examine the whole neighbourhood but only a fraction of it (i.e. best of k instead of best of all) and stop the search when no better neighbour is found after a small number of iterations. Later, they also proposed to apply local search to only good offspring to improve the search ability of their genetic local search approach [35]. In this paper, the best half of each sub population (based on fitness) is chosen to improve by local search. And in every 10 generations, the local search procedure, SA, is run.

3.3. Components of the third stage (Final organization)

3.3.1. Recreate Sub populations and Regrouping

In stage 2, each sub population was isolated and could not share its findings with the other sub populations. In this stage using a process similar to stage 1, the population arrangement is done again. That is, the solutions are sorted in a descending order according to their fitness. Then the entire population is divided into m sub populations, each containing n solutions. Finally, a process similar to the SFL is applied. That is, the first solution goes to the first sub population, the second solution goes to the second sub population, population m goes to the mth sub population, and solution m+1 goes back to the first sub population, etc.

4. Experimental results

The JSFMA was implemented in Matlab and the tests were run on a PC with Pentium IV 2.8 GHz processor and 2 GB memory. In order to give a rough idea about the quality achieved, we confined to the 20 problems of the LA test problems, LA01-LA20, that were reported by Lawrence in 1984 [36]. Applied instances of this data set consist of the problems with 10, 15 or 20 jobs, 5 or 10 machines, and 5 or 10 operations. The benchmark instances considered in the experiments are summarized in Table 2. In the table, the first column shows the instance name and the second one indicates the relevant reference.

Table 2.Benchmark instances

Instance	**Reference**
ft06-ft10- ft20	Muth and Thompson [37]
la01-la40	Lawrence [36]
abz5-abz9	Adams et al. [38]
orb01-orb10	Applegate and Cook [39]
swv01-swv20	Storer et al. [40]
yn1-yn4	Yamada and Nakano [41]

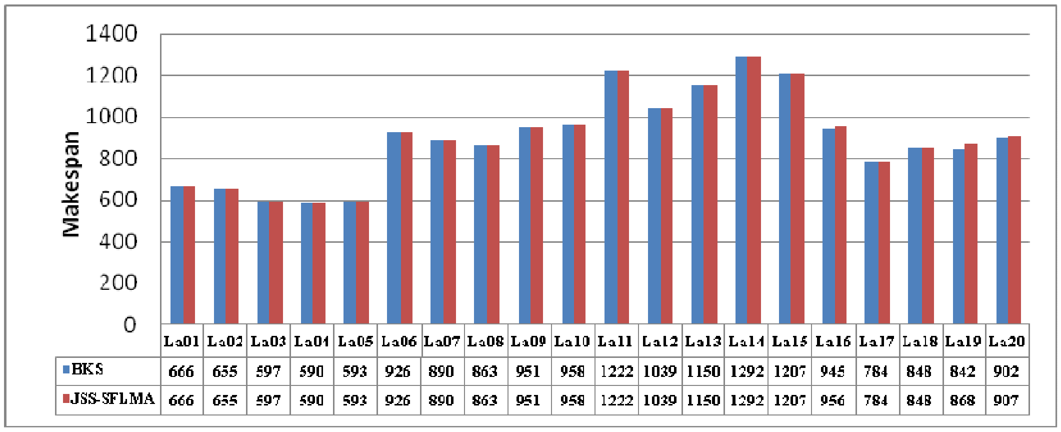
Table 3 summarizes the results of our experiment and compares them with the results of the other literature considered. The contents of the table respectively include the test problem name (INS), the number of jobs (J-no), the number of operations (O-no), the value of the best known solution (BKS), the value of the best solution found by the proposed approach (JSFMA), the relative deviation of this approach with respect to BKS (Dev%), and finally the values of the best solution obtained by Gao et al. [19], Yang et al. [9], Park et al. [42], Nuijten [43], and Coello [44] . The relative deviation is defined as Eq.2.

$$Dev = \left[\frac{MK_{JSFMA-}\ Mk_{BKS}}{MK_{JSFMA}}\right] \times 100 \qquad (2)$$

Where, MK_{JSFMA} is the makespan obtained by the proposed approach and MK_{BKS} is the best known makespan. Results show that in 80% of all cases, the bold numbers shown in column JSFMA of Table 3, our approach could find the BKS successfully. The proposed approach could gain the Best Known Solutions to LA01-LA20 with the exception of LA16, LA19, and LA20. Figure 6 draws the makespan of JSFMA and the best known solution in comparison. As shown in Table 4, in 17 out of 20 problems JSFMA could gain the same good results as the literature. Information of the other specified papers is also shown in Table 4. In that table, the first column refers to the criteria chosen to compare algorithms. The second column shows the instances names the tests were run by them. The remaining columns refer to the values of the proposed approach and the other papers. To show the proposed approach efficiency, average relative deviations of the JSFMA with respect to the several papers are calculated and shown in Table 5.

Table 3.Experimental results

INS	J-no	O-no	BKS	JSFMA	Dev (%)	Gao [19]	Yang [9]	Park [42]	Nuijten[43]	Coello [44]
La01	10	5	666	**666**	0	666	666	666	666	666
La02	10	5	655	**655**	0	655	655	666	666	655
La03	10	5	597	**597**	0	597	597	597	597	597
La04	10	5	590	**590**	0	590	590	590	590	590
La05	10	5	593	**593**	0	593	593	593	593	593
La06	15	5	926	**926**	0	926	926	926	926	926
La07	15	5	890	**890**	0	890	890	890	890	890
La08	15	5	863	**863**	0	863	863	863	863	863
La09	15	5	951	**951**	0	951	951	951	951	951
La10	15	5	958	**958**	0	958	958	958	958	958
La11	20	5	1222	**1222**	0	1222	1222	1222	1222	1222
La12	20	5	1039	**1039**	0	1039	1039	1039	1039	1039
La13	20	5	1150	**1150**	0	1150	1150	1150	1150	1150
La14	20	5	1292	**1292**	0	1292	1292	1292	1292	1292
La15	20	5	1207	**1207**	0	1207	1207	1207	1207	1207
La16	10	10	945	956	-1.16	945	945	977	977	945
La17	10	10	784	**784**	0	784	784	787	787	785
La18	10	10	848	**848**	0	848	848	848	848	848
La19	10	10	842	868	-3.09	842	844	857	848	848
La20	10	10	902	907	-0.55	902	907	910	907	907
Average Dev					-4.8					



	La01	La02	La03	La04	La05	La06	La07	La08	La09	La10	La11	La12	La13	La14	La15	La16	La17	La18	La19	La20
■BKS	666	655	597	590	593	926	890	863	951	958	1222	1039	1150	1292	1207	945	784	848	842	902
■JSS-SFLMA	666	655	597	590	593	926	890	863	951	958	1222	1039	1150	1292	1207	956	784	848	868	907

Figure 6.The comparison between the makespan of JSFMA and BKS (Best Known Solution)

Table 4.The comparison between the proposed approach and the other papers

Criteria	**Instance name**	**JSFMA**	**Gao [19]**	**Yang [9]**	**Park [42]**	**Coello [44]**	**Nuijten [43]**
#BM	LA01-La20	0	0	0	0	0	0
#SM		17	20	18	15	17	15
#WM		3	0	2	5	3	5

The #BM indicates the number of solutions with the better makespan rather than BKS.

The #SM indicates the number of solutions equal to BKS.

The #WM indicates the number of solutions with the worst makespan rather than BKS.

Table5. Average relative deviation of JSFMA with respect to other papers

Instance Name	**Literature**	**Average DEV%**
LA01-LA20	Gao [19]	-4.81
	Yang [9]	-4.01
	Park [42]	3.23
	Coello [44]	1.82
	Nuijten [43]	-3.40

4. CONCLUSION

The JSSP belongs to the NP-hard family of problems and still the optimal solution for all test problems could not be found by the proposed algorithms. Over the last decades a good amount of research has been reported aiming to solve JSSP by means of different algorithms. One of the algorithms used to solve JSSP is the Memetic Algorithm. This effective algorithm keeps the balance of exploration and exploitation.

In this paper, a three-stage multi population based hybrid approach, JSFMA, is proposed to solve JSSP. In fact, we seek solutions to the Job Shop Scheduling Problem by means of a Memetic Algorithm that combines a Genetic Algorithm with a Simulated Annealing based local search. To find solutions three stages are considered. In the first stage parameters setting and the population initialization are done. Then the population is divided into several groups using the SFLA, Shuffled Frog Leaping Algorithm, method. In the second stage the Memetic Algorithm is applied. In the last stage the groups are combined and using SFLA method re-grouping operation is done. In this way the individuals can migrate to other groups. The approach is compared with several algorithms proposed in the literature and the tests are done on a set of 20 standard instances from the Lawrence's dataset. The computational results show that SFLMA can find the Best Known Solution in 80% of the instances, LA01-LA20.

REFERENCES

[1] Zhang, G. & Gao, L & Shi, Y. (2008) "A genetic algorithm and taboo search for solving flexible job shop schedules", in proc. computational intelligence and design International symposium, pp369-372.

[2] Giovanni, L. D. & Pezzella, F. (2010) "An improved genetic algorithm for the distributed and flexible job shop scheduling problem", European journal of operational research, Vol. 200, pp395-408.

[3] Muth, J. F. & Thompson, G. (1963) "Industrial Scheduling", Prentice Hall, Englewood Cliffs, New Jersey, pp225-251.

[4] Fang, H. & Ross, P. & Corne, D. (1993) “A promising genetic algorithm approach to: job shop scheduling, rescheduling and open shop scheduling problems”, fifth international conference on genetic algorithms, pp375-382.

[5] Brucker, P. & Schile, R. (1990) “Job-shop scheduling with multi-purpose machines, Computing, Vol. 45, No. 4, pp369-375.

[6] Huang, W.Q. & Yin, A. H. (2004) “An improved shifting bottleneck procedure for the job shop scheduling problem”, *Computers & Operations Research*, Vol. 31, pp2093–2110.

[7] Chen, H. X. & Luh, P. B. (2003) “An alternative framework to Lagrangian relaxation approach for job shop scheduling”, European Journal of Operational Research, Vol. 149, pp499–512.

[8] Nowicki, E. & Smutnicki, C. (1996) “A fast taboo search algorithm for the job shop problem”, Management Science, Vol. 42, pp797–813.

[9] Yang, J. & Sun, L. & Lee, H. & Qian, Y. & Liang, Y. (2008) “Clonal Selection Based Memetic Algorithm for Job Shop Scheduling Problems”, Journal of Bionic Engineering, Vol. 5, pp111-119.

[10] Moscato P. (1989) “On genetic crossover operators for relative order preservation”, C3P Report 778, California Institute of Technology, Pasadena, CA 91125

[11] Dawkin, R. (1976) *the Selfish Gene*, Clarendon Press, Oxford.

[12] Ishibuchi, H. & Narukawa, K. (2004) “Some issues on the of implementation of local search in evolutionary Multi-objective optimization”, Lecture Notes in Computer Science, Vol. 3102, pp1246-1258

[13] Jaszkiewicz, A. (2004) “A comparative study of multiple-objective meta-heuristics on the bi-objective set Covering problem and the pareto memetic algorithm”, Operations Research, Vol. 131, No. 1-4, pp135-158

[14] Knowles, J.D. & Corne, D.W. (2000) “M-paes: a memetic algorithm for multi-objective optimization”, in the 2000 congress on Evolutionary Computation, San Diego, USA, pp325-332

[15] Moscato, P. & Cotta, C. (2005) “Memetic algorithms”, University of Newcastle, Australia

[16] Krasnogor N. & smith J. (2000) “a memetic algorithm with self adaptive local search: TSP as a case study”, In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Morgan Kaufmann, San Francisco, USA, pp987-994.

[17] Ong, Y. & Lim, M. & Zhu, N. & Wong, K. (2006) “Classification of adaptive Memetic algorithms: A comparative study”, IEEE transactions on systems, man, and cybernetics—part b: cybernetics, Vol. 36, No. 1, pp141-152.

[18] Bergh, F. (2001) “An analysis of particle swarm optimizers”, university of Pretoria.

[19] Gao, L. & Zhang, G. & Zhange, L. & Li, X. (2011) “an efficient memetic algorithm for solving the job shop scheduling problem”, Computers & Industrial Engineering, Vol. 60, pp699–705.

[20] Elbeltagi, E. & hegazy, T. & Grierson, D. (2005) “comparison among five evolutionary-based optimization algorithms”, Advanced Engineering Informatics, pp43-53.

[21] Holland, J.H (1992) “Adaptation in Natural and Artificial Systems”, 2nd Ed, MIT Press

[22] Sivaraj, R. & Ravichandran, T. (2011) “A review of selection methods in Genetic algorithm”, International journal of engineering science and technology, Vol. 3, No. 5, pp3792-3797

[23] Bierwirth, C. (1995) “A generalized permutation approach to job shop scheduling with genetic algorithms”, OR Spectrum, pp1787-1792.

[24] Fogel, D. (1993) “applying evolutionary programming to selected traveling salesman problems”, Cybernetics and systems, Vol. 24, pp27-36

[25] Larranaga, P. & Kuijpers, C.M.H. & Murga, R.H. and Dizdarevic, S. (1999) “genetic algorithms for the travelling salesman problem: a review of representations and operators”, Artificial Intelligence Review, Vol. 13, pp. 129-170.

[26] Metropolis, N. & Rosenbluth, A. W. & Rosenbluth, M. N. & Teller, A. & Teller, E. (1953) “Equation of State Calculation by Fast Computing Machines”, Journal of Chem. Phys, Vol. 21, pp1087-1091.

[27] Kirkpatrick, S. & Gelatt, C. D. & Vecchi, M. P. (1983) “Optimization by Simulated Annealing”, Science, Vol. 220, pp671-680

[28] Kendall G, “http://www.cs.nott.ac.uk/ ~gxk/aim/notes/simulatedannealing.doc”

[29] COPASI development team (2009) “http://www.copasi.org”

[30] Russell, S. & Norvig, P. (1995) “Artificial Intelligence: A Modern Approach”, Prentice Hall, pp113-114.

[31] Ishibuchi, H. & Murata, T. & Yoshida,T. (2003) “Balance between genetic search and local search in Memetic algorithms for multi objective permutation flow shop scheduling”, IEEE transactions on evolutionary computation, Vol. 7, No. 2, pp204-223

[32] Bruke, E. & Smith, A. (1999) “a memetic algorithm to schedule planned maintenance for the national grid”, ACM Journal of experimental algorithmics, Vol. 4, No. 1, pp1084-1096

[33] Burke, E. K. & Landa Silva, J. D. (2004) “The Design of Memetic Algorithms for Scheduling and Timetabling Problems”, Krasnogor N., Hart W., Smith J. (eds.), *Recent Advances in Memetic Algorithms*, Studies in Fuzziness and Soft Computing, Springer, vol. 166, pp289 – 312.

[34] Ishibuchi, H. & Murata, T. & Tomioka, S. (1997) “effectiveness of genetic local searchalgorithms”, proceedings of the seventh international conference on genetic algorithms, pp505-512.

[35] Ishibuchi, H. & Yoshida, T. & Murata, T. (2002) “selection of initial solutions for local search in multiobjective genetic local search”, proceeding of the 2002 congress on evolutionary computation (CEC 2002), pp950-955.

[36] Lawrence, S. (1984) “Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques”, Pittsburgh, PA: GSIA, Carnegie Mellon University.

[37] Muth, J. F. & Thompson, G. L.(1963) *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, New Jersey, pp. 225-251

[38] Adams, J. & Balas, E. & Zawack, D. (1988) “The shifting bottleneck procedure for job shop scheduling”, Management Science, Vol. 34, pp391-401

[39] Applegate, D. & Cook, W. (1991) “A computational study of the job-shop scheduling instance”, ORSA Journal on Computing, Vol. 3, pp149-156

[40] Storer, R.H. & Wu, S.D. & Vaccari, R. (1992) “New search spaces for sequencing instances with application to job shop scheduling”, Management Science, Vol. 38, pp1495-1509

[41] Yamada, T. & Nakano, R. (1992) “A genetic algorithm applicable to large-scale job-shop instances”, North-Holland, Amsterdam, pp281-290

[42] Park, B. J. & Choi, H. R. & Kim, H. S. (2003) “A hybrid genetic algorithm for the job shop scheduling problems”, Journal of Computers & Industrial Engineering, Vol. 45, No. 4, pp597–613.

[43] Nuijten, W.P.W. & Aarts, E. H. L. (1996) “computational study of constraint satisfaction for multiple capacitated job shop scheduling”, European journal of operational research, Vol. 90, pp269-284

[44] Coello, C. A. C. & Rivera, D. C. & Cortes, N. C. (2003) “use of an artificial immune system for job shop scheduling”, lecture notes in computer science, Vol. 2787, pp1-10