Journal of Industrial Engineering 4 (2009) 71- 78

A Cellular Learning Automata (CLA) Approach to Job Shop Scheduling Problem

Masoud Abdolzadeh[a], Hassan Rashidi[a,*]

[a]*Computer Engineering Department, Islamic Azad University, Qazvin Branch, Qazvin, Iran*

Received 15 Sep., 2009; Revised 5 Oct., 2009; Accepted 19 Oct., 2009

Abstract

Job shop scheduling problem (JSSP), as one of the NP-Hard combinatorial optimization problems, has attracted the attention of many researchers during the last four decades. The overall purpose regarding this problem is to minimize maximum completion time of jobs, known as makespan. This paper addresses an approach to evolving Cellular Learning Automata (CLA) in order to enable it to solve the JSSP by minimizing the makespan. This approach is applied to several instances of a variety of benchmarks and the experimental results show that it produces nearly optimal solutions, compared with other approaches.

Keywords: Job Shop; Scheduling; Makespan; Cellular Learning Automata.

1. Introduction

Scheduling problems in various systems is one of the major challenges to reach high performance. In order to simplify algorithms presentation and analyze real world problems, scheduling problems are classified into different groups. Each real world problem is assigned to one of these groups and solved with appropriate solutions. One of these problems is called Job Shop Scheduling Problem (JSSP). This problem includes resource assigning to a set of operations in their given time [2]. Industrial tasks ranging from assembling cars to scheduling airplane maintenance crews are easily modeled as instances of this problem, and improving solutions by even as little as one percent can have a significant financial impact. Furthermore, this problem is interesting from a theoretical standpoint as one of the most difficult NP-Hard problems to solve in practice and only small groups of them can be solved by searching all problem space [3].

A typical problem of JSSP with m machines and n Jobs has $(n!)^m$ possible states in its search space. Thus for problem with 10 jobs and 10 machines there are $7.2651 * 10^{183}$ possible states [9].

The approaches in literature to scheduling problem are classifies in two groups. The first group identifies an exact solution via integer programming, dynamic programming, branch and bound (B&B) and enumeration [19]. this group needs too much time and becomes inefficient as the number of jobs increases. The second group is meta-heuristic, which yields an approximate solution in a very short time; the methods involved in the group are very diverse, including a Genetic Algorithm (GA), Simulated Annealing (SA), Tabu Search (TS), Ant Colony Optimization (ACO), and Neural Networks (NN) [1,4,5,6,7,10,14,15,17,20,21]. These algorithms use problem search space and try to optimize the first or the current solutions and repeat optimization to get terminate criterion.

We propose an approach to JSSP based on the cellular learning automate. There is no solution for the problem by this approach. At the end-quarter of the 20th century, Cellular Automata was proposed as a model to analyze treatments of complex systems. Learning Automata was presented at the beginning of 1960s which treats based on learning algorithm. This model learns how to choose its best action from a set of actions. The Cellular Learning Automaton proposed was based on a combination of cellular and learning automata. In this model, each cell is equipped with a learning automaton that determines cell's state, which is a specific solution. This paper optimizes the solutions to the JSSP using features of CLA and makes possible learning the position of jobs in job sequence.

The rest of this paper is organized as follows: In section 2, literature review of the JSSP is given. A detailed description of the JSSP follows in section 3. Section 4 summarizes the cellular learning automata. In

*Corresponding author E-mail: hrashi@essex.ac.uk

section 5, our proposed algorithm is described. Section 6 is considered for reporting experimental results and their subsequent comparison with other algorithms. Section 6 presents the conclusion and rounds up the paper.

2. Literature Review

Job shop makespan minimization is a challenging problem. However, efficient, polynomial time solutions have been found for the problem. Work on optimal solution procedures for general makespan problems has focused most heavily on the development of implicit enumeration, or branch and bound approaches. These approaches were not able to solve larger problems. As observed in [12], the applicability of implicit enumeration schemes is limited to relatively small problems, and their performance is quite sensitive to particular problem instances and initial upper bound values.

Other research studies have investigated heuristic approaches. Simple priority dispatching algorithms are the most representative and the most widely used in practical environments. While dispatching algorithms are extremely fast and easy to implement, there are also drawbacks. The performance of any given rule is typically quite sensitive to problem characteristics, and it is generally quite difficult to find one that dominates in any particular environment. Such procedures are also susceptible to very poor performance in certain circumstances, due to the myopic nature of their decision-making. With the rapid increase in computing power in recent years, a growing body of research has focused on development of more sophisticated heuristic methods, which incorporate various forms of search and aim at striking a better cost performance tradeoff than the extremes that are provided by dispatch and optimal solution procedures. One such notable approach emphasizes bottleneck tracking as a heuristic methodology for integrating optimal solutions to simpler, one-machine sub problems. A series of shifting bottleneck procedures have been defined which have demonstrated very strong performance on a range of previously published benchmarks, and provide a continuum of increasingly more accurate solution procedures at increasing computational expenses.

Other work has explored the use of various local search techniques as a basis for approximate solution which yields a solution in a very short time; the methods involved in the work are very diverse, including a Genetic Algorithm (GA), Simulated Annealing (SA), Tabu Search (TS), Ant Colony Optimization (ACO), and Neural Networks (NN) [1,4,5,6,7,10,14,15,17,20,21]. These techniques use problem search space and try to optimize the first or the current solutions and repeat optimization to get terminate criterion.

3. Job Shop Scheduling Problem

In this section, we define JSSP, describe the problem, and provide solution representation methods.

3.1. Problem Definition

A JSSP can be defined by (n) jobs and (m) machines. Each job consists of several operations. Each operation should be processed by specified machine. Processing time for each operation is fixed and predefined. In other words, there is a sequel of machines proportionate to each job that must be processed. We suppose all jobs are ready at the beginning time. Initialization time of operations is set to zero or as a part of processing time. There is no precedence between jobs. Each machine can process just one operation of a job and each job can be processed by one machine at a time. There is no permission to interrupt for operation processing.

We can define construction of JSSP as follows:

- A set of N independent Jobs. $\{J_j\}$ $1 \leq j \leq N$
- Each J_j has a sequence of operations. (G_j)
- Each G_j is ordered as series of operations and $O_{i,j}$ determines the position of an operation in the job sequence. There is precedence between the operations of a job. In other words, operation $O_{i+1,j}$ cannot be processed before operation $O_{i,j}$.
- Each $O_{i,j}$ needs to execute on a specific machine from the machine set : $\{M_k\}$ $1 \leq k \leq m$ (m is the number of machines). This issue implicitly makes an assignment problem.
- There is a set of predefined processing time. $P_{i,j}$ is the processing time of $O_{i,j}$.
- Each machine processes operations sequentially.
- C_{max} is the completion time of all jobs (makespan).

There are infinite buffers to queue operations before and after each machine. Each possible scheduling must determine start time of operations. Thus there is no conflict between operations. The goal in the JSSP is to find possible solutions that determine processing order of operations and processing capacity of machines. In other words, it identifies the set of finishing time for each operation $\{C_{i,j,k}\}$ and minimizes makespan.

3.2. Problem Representation

A table with n rows and m columns can be used to represent a JSSP. Rows and columns present jobs and the corresponding machines for their operations, respectively. Each cell of the table contains machine number and its processing time. Table 1 shows a typical JSSP with 2 jobs and 3 machines.

Table 1
A JSSP with 2 jobs and 3 machines

Job	Machine(processing time)		
1	1(3)	2(2)	3(5)
2	3(3)	2(5)	1(4)

Another way to represent JSSP is using of disjunction graph $G(V,C \cup D)$. V is the set of nodes. In this set, there is one node per each operation. Also, there are two beginning (O) and finishing (*) nodes in this set. These nodes distinguish start and stop time of scheduling, respectively. C is the set of conjunction directional edges of graph. The Members of C are used to present processing order of operations. D is the set of disjunction (bidirectional) edges. There is a disjunction edge between each pair of operations on a machine in corresponding graph. The Processing time of each operation is assigned to corresponding node as weight. Figure 1 shows a graphical schema of Table 1.

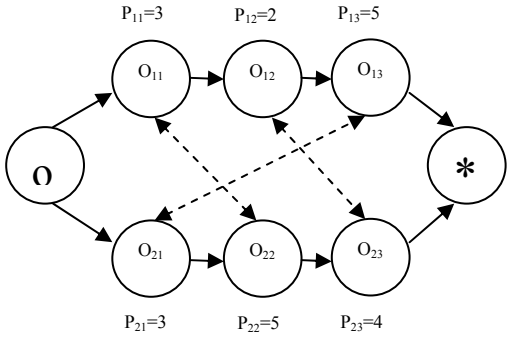

Fig. 1. Graphical representation of Table 1.

O_{ij} is the j^{th} operation of job i and P_{ij} is the processing time of j^{th} operation of job i. The two nodes (O) and (*) are beginning and finishing nodes which determine start and stop time of scheduling, respectively. JSSP solutions can be shown in various methods. Three common methods to represent JSSP solutions are as follows:

1. *Disjunction graph:* By determining the processing order of machines, a scheduling can be defined. If we convert disjunction edges (bidirectional edges) to directional edges, the obtained graph shows a possible scheduling solution. This graph should not have any cycle.
2. *Permutation:* A schedule is represented by a set of permutations of jobs on each machine. In other words, there are m partitioned permutations of operation numbers. A possible representation for a 3×3 problem could be:

M_1	M_2	M_3
132	231	213

3. *Permutation with repetition:* By repeating jobs for m times (m is the number of machines), an un-partitioned operation based representation can be used. The k^{th} occurrence of a job number would refer to k^{th} operation of that job. The sequence of operations, thus, represents the priority in which they are to be scheduled. We can produce a possible schedule by scanning job numbers from left to right. Figure 2 shows a possible representation for a 3×3 problem. The third number, for instance, shows the second operation of job 2 (O_{22}) that must be executed on machine M_3 after first operation on the same job (O_{21}). The next number is the first operation of job 3 (O_{31}) that must be executed on machine M_2 after O_{21}.

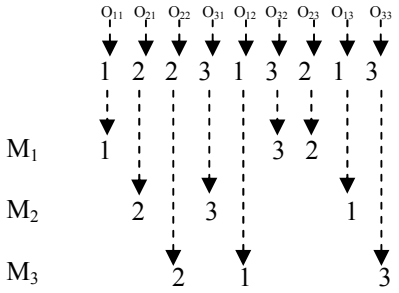

Fig. 2. The un-partitioned permutation representation of 3×3 JSSP problem

Different representation methods affect the used algorithms. Thereby, the chosen presentation method must be compatible with algorithm operators and should not cause to produce impossible solutions. If so, the cost of converting impossible solution to possible solution should not be too much. We used permutation with repetition representation method in the algorithm. The second method (permutation) can be useful too, but, as the occurrence of deadlock is possible in this method, we must use a way to eliminate deadlock. Because this elimination makes additional costs, the second method is not useful in algorithm. In permutation with repetition representation method, the jobs are scanned from left to right order and then just one job can be chosen at a time, but in the second method choosing m jobs is possible. For instance, a typical representation for a 2*2 problem by permutation method is as follows:

M_1	M_2
O_{12}, O_{21}	O_{22}, O_{11}

We suppose each machine processes operations from left to right order, and, then, O_{11} and O_{21} should be processed after O_{12} and O_{22}, respectively. On the other hand, O_{11} and O_{21} are prior to O_{12} and O_{22}, respectively. Then, in this example, none of the operations can be processed on the machines. Thus, sometimes deadlock occurrence is not avoidable. One way to eliminate deadlock is generating new permutation for a randomly chosen machine.

4. Cellular Learning Automata

In this section, we briefly describe Learning Automata, Cellular Automata and Cellular Learning Automata.

4.1. Learning Automata

Learning automata is a machine that can execute a finite set of actions [18]. Each action has a certain probability (unknown for the automaton) and is evaluated by the environment. The results of evaluation are sent to automaton as a positive or negative signal. Each action gets reward by this signal. The aim is to learn to choose the optimal action (i.e. the action with the highest probability of being rewarded) through repeated interaction on the system. Figure 3 illustrates how a learning automata works in feedback connection with a random environment.

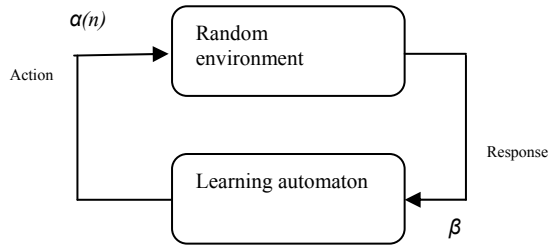

Fig. 3. Interaction between environment and learning automaton.

An Environment is a triple $E=\{\alpha, \beta, c\}$, where $\alpha=\{\alpha_1, \alpha_2 ,..., \alpha_s\}$, $\beta=\{\beta_1, \beta_2, ..., \beta_m\}$, $c= \{c_1,c_2,...,c_s\}$ are an action set with s actions, an environment response set and the penalty probabilities set c containing s probabilities, respectively. Each element c_i of c corresponds to one input of action α_i. Learning automata generates actions $\alpha(n)$. These actions are evaluated by environment. The environment response (β) is sent to learning automata to generate the next generation of actions $\alpha(n+1)$. There are three types of environment based on its response. Whenever $\beta=\{\beta_1,\beta_2\}$ is a two members set, environment type is P. In this environment, $\beta_1=1$ is the punishment and $\beta_2=0$ is the reward. A further generalization of the environment, known as Q-model, allows finite output set with more than two elements that take values in the range [0,1]. A further step in this direction is the S-model whose responses can take continuous values over the unit range [0,1].

Learning Automata can be classified into two main families: Fixed Structure Learning Automata and Variable Structure Learning Automata (VSLA). We consider a VSLA in our algorithm and it operates as follows:

A VSLA is a quintuple $<\alpha,\beta,p,T>$ where α and β are the same parameters as the environment's, and $p=\{p_1,p_2,...p_s\}$ is the action probability set, each being the probability of performing every action in the current internal automaton state. The function $P(n+1)=T[\alpha(n),\beta(n),p(n)]$ is the reinforcement algorithm which modifies the action probability vector p with respect to the performed action and received response at the next generation. This automaton operates as follows. Based on the action probability set p, automaton randomly selects an action α_i, and performs it on the environment. After receiving the environment's reinforcement signal, automaton updates its action probability set based on the following reinforcement scheme; the equations (1) for favorable response, and equations (2) for unfavorable one. When an action gets reward, its probability p_i increases, while the probability of all other actions decreases.

If $\beta=0$ (1)

$$P_i(n+1) = P_i(n) + a[1 - P_i(n)]$$
$$P_j(n+1) = (1-a)P_j(n) \forall j, j \neq i$$

If $\beta=1$ (2)

$$P_i(n+1) = (1-b)P_i(n)$$
$$P_j(n+1) = (b/r-1) + (1-b)P_j(n) \forall j, j \neq i$$

Where a and b are reward and penalty parameters. When $a=b$, automaton is called "L_{RP}". If $b=0$ and $0<b<<a<1$, the automaton is called "L_{RI}" and "$L_{R\varepsilon p}$", respectively [8].

4.2. Cellular Automata

Cellular Automata consist of a regular array of cells, each in one of finite number of states. The array can be in any finite number of dimensions. The state of a cell at the time $t+1$ in cellular automata is a function of the states of a finite number of cells (called its "neighborhood") at the time t. The simplest cellular automata would be one dimensional with two possible states per cell and a cell's neighbors defined to be the adjacent cells on either side of it.

4.3. Cellular Learning Automata

CLA is obtained by combining cellular automata and learning automata. It is a mathematical model for dynamical complex systems that consists of a large number of simple learning components. Any number of learning automaton can reside in a specific cell. Reinforcement signal for every automaton is computed according to CLA rule and actions of other learning automata residing in neighbor cells. This model has learning capability of learning automata and collective behavior and locality of cellular automata. A d dimensional CLA is a quintuple $CLA=(Z^d ,Ø,A,N,F)$ that :

- Z^d is a d-dimensional grid of cells.
- $Ø$ is a finite set of states that each cell can possess.
- A is set of learning automata that each of them are assigned to a specific cell.
- $N = \{X_1 ,...,Xm \}$ is finite subset of Z^d that is called neighborhood vector.
- $F: Ø^m \rightarrow \beta$ is local rule of CLA. β is a set of valid reinforcement signals that can be applied to learning automata.

Like cellular automata, CLA operate subject to a rule. The rule of CLA and the actions selected by neighboring learning automata of any particular learning automaton determine the reinforcement signal to the learning automaton residing in that cell. In CLA, the neighboring learning automata of any particular learning automaton constitute its local environment, which is non-stationary because it varies as action probability vector of neighboring learning automata varies.

5. The Proposed Algorithm

In this section, a new approach to solving Job Shop Scheduling Problem, using cellular learning automata is proposed. To this end, first, a solution representation method should be chosen. In this paper, permutation with repetition is used. We used an array with a length of m*n to represent solutions named Cell. Each cell has m*n learning automata. Figure 4 shows the construction of used CLA in our algorithm.

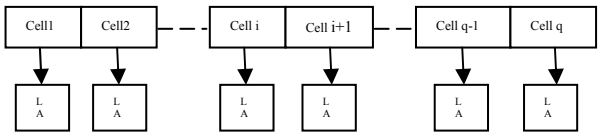

Fig. 4. The construction of CLA in the proposed algorithm

Where *LA* is the learning automata residing in each cells. We consider two neighborhoods for each cell. Thus, for cell *i*, neighbors are cell *(i-1)* and cell *(i+1)*. Each cell chooses its best neighbor based on makespan values. In other words, the best neighbor for one cell is the cell that has the lowest makespan value denoted here as LB_i. Furthermore, the best global cell based on makespan value is denoted as *GB* and the best previous cell state set to *PB* array.

Since the JSSP is the ordering problem and each element of the cell represents an operation of jobs, thus, for every learning automaton, actions 'set $\alpha(n)$ must be chosen based on this purpose.

Two kinds of actions' sets can be used in this paper:

- In a form of a set with two actions as follows:
  a) Preserve current position of O_{ij} in the cell.
  b) Move O_{ij} to other position.
- In a form of a set with n actions (*n* is the number of jobs) that each action (α_i) is "choose operation (*i*)".

In the first method, the probability set has two members for each learning automata which is initialized with 0.5 values for each action. Actions are chosen based on corresponding probability. If the cell has a lower makespan value in the next generation, we reward actions whose corresponding operation positions (O_{ij}) are suitable and try to preserve their position in the cell. Otherwise, by selecting the other action and changing position of O_{ij} the other states are checked.

In the second method, probability set has *n* members for each learning automata which is initialized with *1/n* values for each action. Like above, the operations of jobs with suitable positions are rewarded and by increasing their corresponding action probability, situation preserving chance is grown. Otherwise, by decreasing corresponding probability and increasing the other jobs' probabilities, possibility of choosing job's operation in the next generation is weakened.

In the first method, to prevent convergence and local minimums, we forced the probabilities of actions lie in the range of $[1-P_{max}, P_{max}]$. P_{max} is parameter with $P_{max}>0.5$ values. If the actions probabilities exceed from this range, we convert them to this range.

In the second method, to prevent premature convergence and local minimums, we used an operator similar to the mutation operator in genetic algorithm. If cells stay in one state for several generations, we change some elements' position randomly.

Reinforcement signal is considered as $\beta=\{0,1\}$. If reward signal is generated for environment, $\beta_{ij}=0$. Otherwise, if punishment signal is generated, $\beta_{ij}=1$. This approach uses learning automata with type L_{RP}.

The stages of our proposed algorithm are as follows:

1. First generate initial cells with action probabilities equal to *1/k* values (*k* is the number of actions) for each learning automaton. Each cell has *m*n* learning automata.
2. Generate actions based on probabilities vector for each learning automata.
3. Generate new cells (X_i) using learning automata.
4. Obtain makespan value for the new cell and compare it with PB_i. If the new cell has a better makespan value, replace it. (update PB_i)
5. Choose the best neighbor (LB_i) for each cell based on makespan values.
6. Choose the best global cell too (*GB*).
7. According to the following conditions generate reinforcement signal for each learning automaton and update probability vectors.

   If ($X_i.C_{max}<=LB_i.C_{max}$ && $X_i.C_{max}<=PB_i.C_{max}$ && $X_i.C_{max}<=GB.C_{max}$)
   { *For each j residing in cell i*
   If($X_{ij}==PB_{ij}$ && $X_{ij}==LB_{ij}$ && $X_{ij}==GB_j$)
   Reward X_{ij}
   Else
   Penalt X_{ij}
   }

Where X_{ij}, PB_{ij} and LB_{ij} are the j^{th} elements of X_i, PB_i and LB_i, respectively. C_{max} is the makespan value. In the aforementioned algorithm, reinforcement signal is generated for each cell based on current and previous makespan values and the neighbor cell' states. Then, the

probability vectors are updated and a new generation of the cells is produced. This algorithm is applied on all cells simultaneously and will be repeated until a termination criterion is met.

6. Simulation Results

We used *C#* language to implement our algorithm and perform a simulation. The Simulation is executed on PC platform with configuration of:

- 1.86 GHz CPU,
- 1MB RAM and
- Windows XP operating system.

In order to analyze performance of the algorithm, it was tested on several instances of some benchmarks such as ABZ7, ABZ8, LA19, LA21, LA22, LA24, LA38 and LA40. These benchmarks are shown on Table 2. In this simulation, we used CLA with type of L_{RP} and with reward and penalty rates $\alpha=\beta=0.1$.

The algorithm is repeated on all the mentioned benchmarks until a termination criterion is met. In order to show convergence process, we took makespan values of a randomly chosen benchmark LA19 along the 1000 iterations with 100 initial cells. Figure 5 illustrates the convergence process of the algorithm on LA19 benchmark.

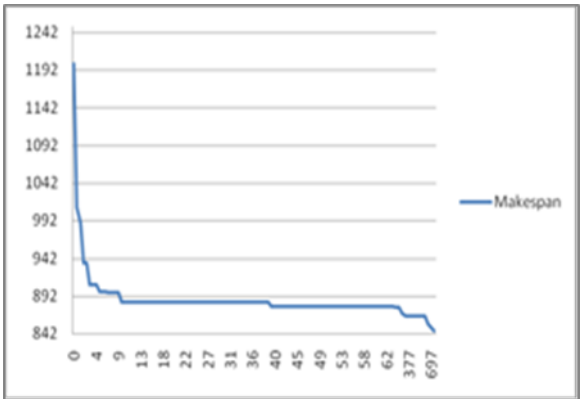

Fig. 5. Convergence process of the simulated algorithm on LA19 benchmark

The vertical axis shows makespan values and horizontal axis represents the spent CPU time. In the beginning of the optimization, the best makespan value is 1223. After 1000 iterations, the best obtained makespan value is 845. We see that the makespan is significantly decreasing over the execution time. The convergence rate is higher initially but it reduces as we proceed to the next iterations. This process continues until the solution converges to the optimal solution, 842.

The rates of reward and penalty (α and β) affects the premature convergence rate. We tested the algorithm on LA19 benchmark with three different reward and penalty rates. Figure 6 shows convergence process of the algorithm on three rates α=β=0.1, α=β=0.3 and α=β=0.5 during 500 iteration.

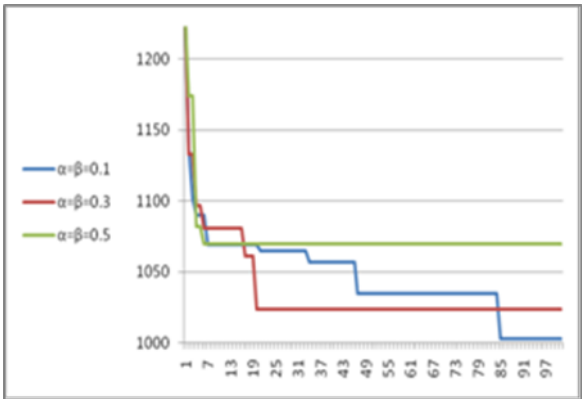

Fig. 6. Convergence process of the simulated algorithm on LA19 benchmark with three different reward and penalty rates

The horizontal axis shows the optimization process and the vertical axis represents makespan values. Figure 6 shows that the premature convergence rate is significantly higher when the reward and penalty rates possess higher values.

The results of simulation are compared with some other approaches. These approaches are: Electro Magnetism, Tabu search with neighborhood structure known as N6, T12 tabu list method and 4628 moves and Genetic algorithm with simple operators, 100 individuals and 3000 generations that have represented good results. Table 2 illustrates the experimental results of the approaches on the mentioned benchmarks. The comparisons in Table 2 are graphically shown in Figure 7. For more information about implementation details and parameters' values of the so-called approaches, refer to [7, 16, 11, 13].A comparisons of percentage of errors among approaches are calculated in Table 3. As the table depicts, our algorithm most often performs better than the others.

Table 2

Simulation results of solving JSSP using CLA

benchmark	Problem size (n×m)	Optimal value (makespan)	Electro Magnetism	Tabu Search	Genetic Algorithm	Our algorithm (Best)	Our algorithm (worst)	Our algorithm (mean)	S.d	CPU Time Used
Abz7	20×15	656	744	696	700	696	726	714.22	11.007	1617
Abz8	20×15	638	798	697	720	716	745	726.81	10.63	1330
LA19	10×10	842	886	860	850	845	866	855	6.86	314
LA21	15×10	1046	1143	1099	1074	1092	1119	1106	13.81	125
LA22	15×10	927	962	962	940	934	963	946.28	9.93	192
La24	15×10	935	1006	989	984	972	992	985.82	7.31	99
LA38	15×15	1196	1381	1254	1273	1264	1309	1284.33	14.86	516
LA40	15×15	1222	1425	1261	1278	1259	1275	1269.08	9.43	469

Table 3

A comparison of percentages of error among solution results

benchmark	Problem size (n×m)	Percentage error			
		Electro Magnetism	Tabu Search	Genetic Algorithm	Our algorithm (Best)
Abz7	20×15	13.41463	6.097561	6.707317	6.097561
Abz8	20×15	25.07837	9.247649	12.85266	12.22571
LA19	10×10	5.225653	2.137767	0.950119	0.356295
LA21	15×10	9.273423	5.066922	2.676864	4.397706
LA22	15×10	3.77562	3.77562	1.402373	0.755124
La24	15×10	7.593583	5.775401	5.240642	3.957219
LA38	15×15	15.46823	4.849498	6.438127	5.685619
LA40	15×15	16.61211	3.191489	4.582651	3.027823

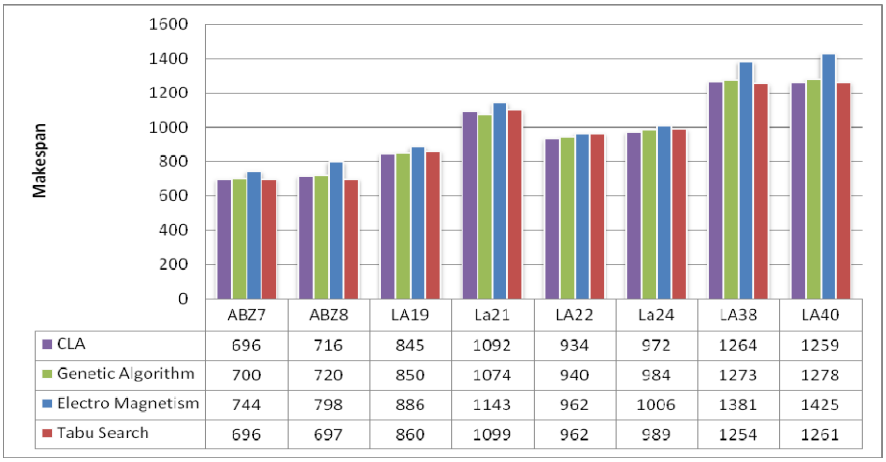

Fig. 7. Results of comparison of the proposed algorithm with other approaches

The horizontal axis shows the benchmarks with detailed corresponding makespan values for each and the vertical axis represents makespan values. The Figure 7 shows that our proposed algorithm often produces solutions with lower makespan values than the other approaches. These values are approach the results of genetic algorithm.

7. Conclusion

In this paper, we used Cellular Learning Automata based approach to solve the Job Shop Scheduling Problem. The approach was presented in a new algorithmic form embracing two kinds of actions' sets. These actions were generated to transfer cells into the best states by changing positions of operations of the jobs. The algorithm could determine the best position of jobs' operation in the execution sequence using the learning aspect of CLA. The results of simulation on several instances of the problem showed that our proposed algorithm often produces optimal or near to optimal solutions. According to our results, the obtained makespan values were significantly better than the other approaches.

References

[1] M. E. Aydin, T. C. Fogarty, A simulated annealing Algorithm for multi-agent systems: a job shop scheduling application. Journal of Intelligent Manufacturing, 15, Netherland, 805-814, 2004.

[2] K. R. Baker, Introduction to sequencing and scheduling, John Wiley and sons Inc, New York, 1974.

[3] J. Carlier, E. Pison, An algorithm for solving the job shop problem. The Institute of Management Science, Vol. 35, U.S.A, 164-176, 1989.

[4] C. Cheng-Chung, S. F. Smith, Applying constraint satisfaction techniques to job shop scheduling. Annals of Operations Research, Springer, Volumn 70, 327 – 357, 1997.

[5] F. L. Cheng-Fa, A New Hybrid Heuristic Technique for Solving Job-shop Scheduling Problem. Technology and Applications, IEEE, 2003.

[6] P. Fattahi, M. Saidi-Mehrabad, F. Jolai, Mathematical modelling and heuristic approaches to flexible job shop scheduling problems. Journal of Intelligent Manufacturing, Springer, Vol. 18, Number 3, 2007.

[7] F. GEYIK, I. Hakki-Cedimoglu, The strategies and parameters of tabu search technique for job-shop scheduling. Journal of Intelligent Manufacturing, Springer, Vol 15, Number 4, 2004.

[8] B. Jafarpour, M. R. Meybodi, S. Shiry, A Hybrid Method for Optimization (Discrete PSO + CLA). International Conference on Intelligent and Advanced Systems, IEEE, 2007.

[9] A. Jain, S. Meeran, Deterministic job-shop scheduling: past, present and future. European Journal of Operational Research, Vol. 113, 390-434, 1999.

[10] S. K. A. C. Kahraman, Meta heuristic Techniques for Job Shop Scheduling Problem and a Fuzzy Ant Colony Optimization Algorithm. Springer, Vol. 201, Berlin Heidelberg, 2006.

[11] T. Kun, H. Zhifeng, C. Ming, PSO with Improved Strategy and Topology for Job Shop Scheduling. Springer Verlag Berlin Heidelberg, 2006.

[12] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy-Kan, D. B. Shmoys, Sequencing and scheduling: Algorithms and complexity. Technical Report, Report Centre Mathematics and Computer Science, Amsterdam, 1989.

[13] D. S. Leea, V. S. Vassiliadisb, J. M. Park, A novel threshold accepting meta-heuristic for the job-shop scheduling problem. Computers & Operations Research 31, Elsevier, 2199–2213, 2004.

[14] H. Rong-Hwa, Y. Chang-Lin, Ant colony system for job shop scheduling with time windows. The International Journal of Advanced Manufacturing Technology, Springer, Vol. 39, Numbers 1-2, London, 2007.

[15] M. Saidi-Mehrabad, P. Fattahi, Flexible job shop scheduling with tabu search algorithms. The International Journal of Advanced Manufacturing Technology, Springer, Volume 32, Numbers 5-6, 2006.

[16] S. M. THashemi, P. Jahanbazi, An Electro magnetism method for job shop scheduling problem with makespan by improving lower bound. Computer society of Iran, 2007.

[17] Y. Takeshi, N. Ryohei, Genetic Algorithms for Job-Shop Scheduling Problems. Proceedings of Modern Heuristic for Decision Support, UNICOM seminar, London, 67-81, 1997.

[18] M. A. L. Thathachar, P. S. Sastry, Varieties of Learning Automata: An overview. Transaction on Systems, Man and Cybernetics-Part B: Cybernetics, IEEE, Vol.32, Number 6, 711-722, 2002.

[19] G. R. Weckman, C. V. Ganduri, D. A. Koonce, A neural network job-shop scheduler. Journal of Intelligent Manufacturing, Springer, Vol 19, Number 2, 2007.

[20] J. C. Werner, M. E. Aydin, Fogarty, T.C., Evolving genetic algorithm for Job Shop Scheduling problems. Proceedings of ACDM, PEDC, University of Plymouth, UK, 2000.

[21] S. V. Zwaan, C. Marques, Ant Colony optimisation for Job Shop Scheduling. ISR– Instituto de Sistemas e Robótica Instituto Superior Técnico (IST), 1998.