Mathematical and Computer Modelling 52 (2010) 1957–1965

Contents lists available at ScienceDirect

Mathematical and Computer Modelling

journal homepage: www.elsevier.com/locate/mcm

An agent-based parallel approach for the job shop scheduling problem with genetic algorithms

Leila Asadzadeh *, Kamran Zamanifar

Department of Computer Engineering, University of Isfahan, Isfahan, Iran

ARTICLE INFO

Article history:
Received 23 September 2009
Received in revised form 25 December 2009
Accepted 31 January 2010

Keywords:
Job shop scheduling problem
Parallel genetic algorithms
Agents
Multi-agent systems

ABSTRACT

The job shop scheduling problem is one of the most important and complicated problems in machine scheduling. This problem is characterized as NP-hard. The high complexity of the problem makes it hard to find the optimal solution within reasonable time in most cases. Hence searching for approximate solutions in polynomial time instead of exact solutions at high cost is preferred for difficult instances of the problem. Meta-heuristic methods such as genetic algorithms are widely applied to find optimal or near-optimal solutions for the job shop scheduling problem. Parallelizing the genetic algorithms is one of the best approaches that can be used to enhance the performance of these algorithms. In this paper, we propose an agent-based parallel approach for the problem in which creating the initial population and parallelizing the genetic algorithm are carried out in an agent-based manner. Benchmark instances are used to investigate the performance of the proposed approach. The results show that this approach improves the efficiency.

1. Introduction

The job shop scheduling problem is one of the most important problems in machine scheduling, and it is an important task for the manufacturing industry in terms of improving machine utilization and reducing product cycle-times. This problem can be described as follows. A set of n jobs $\{J_i\}_{1\leq i\leq n}$ and a set of m machines $\{M_r\}_{1\leq r\leq m}$ are given. Each job consists of a sequence of m_i operations $O_{i1}, O_{i2}, \ldots, O_{im_i}$ that must be processed in a specified order. O_{ij} is the jth operation of job i that must be processed on machine M_k for a processing time period t_{ik}. For each operation O_{ij}, M_k and t_{ik} are predefined and preemption is not allowed. Each machine can process only one job and each job can be processed by only one machine at a time. The duration in which all operations for all jobs are completed is referred to as the makespan. A schedule determines the execution sequence of all operations for all jobs on machines. The objective is to find the optimal schedule. The optimal schedule is the schedule that minimizes the makespan. Due to factorial explosion of possible solutions, job shop scheduling problems are considered to be a member of a large class of intractable numerical problems known as NP-hard [1]. The high complexity of the problem makes it hard and in some cases impossible to find the optimal solution within reasonable time. Hence, searching for approximate solutions in polynomial time instead of exact solutions at high cost is preferred for difficult instances of the problem.

Historically, the job shop scheduling problem has been primarily treated using the branch and bound method [2–4], heuristic rules [5–7] and the shifting bottleneck procedure [8]. In recent years, meta-heuristic methods have been widely applied to this problem. These methods, such as taboo search [9–11], simulated annealing [12–15], genetic algorithms [16–20], neural networks [21] and ant colony optimization [22–25], are well suited to solving complex problems with high costs. A survey of job shop scheduling techniques can be found in [1].

Compared with other meta-heuristic methods, genetic algorithms are widely used to find the best or near-best solutions for various optimization problems. Many genetic algorithm-based approaches have been proposed for job shop

* Corresponding author.
E-mail addresses: leila_asadzadeh_cs@yahoo.com (L. Asadzadeh), zamanifar@eng.ui.ac.ir (K. Zamanifar).

0895-7177/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mcm.2010.04.019

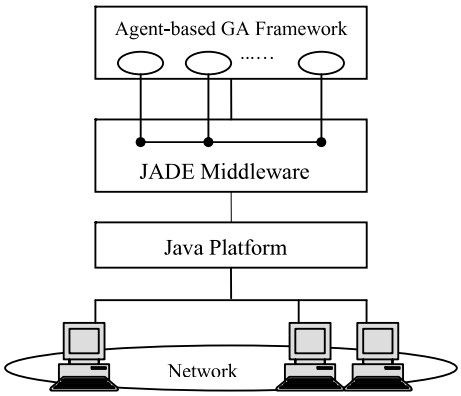

Fig. 1. The layers of the proposed agent-based architecture.

scheduling problems. In [26,27], the authors introduce an approach that uses load balancing of machines as an important parameter in job assignment. An advantage of these approaches is that they maximize machine utilization while minimizing the makespan. Ombuki and Ventresca [28] proposed a local search genetic algorithm that uses an efficient solution representation strategy in which both checking of the constraints and repair mechanism can be avoided. In their approach, at the local search phase a new mutation-like operator is used to improve the solution quality. They also developed a hybrid strategy using the genetic algorithm reinforced with a taboo search for the problem. Lin et al. [29] introduced a hybrid model consisting of coarse-grain genetic algorithms connected in a fine-grain style topology. Their method can avoid premature convergence, and it produced excellent results on standard benchmark job shop scheduling problems. Meta-heuristic methods such as neural networks, simulated annealing and local search were used with genetic algorithms to provide high performance in finding solutions for the problem [17,19,20]. Chen et al. [30] gave a tutorial survey of recent works on various hybrid approaches of the genetic algorithms proposed so far for the job shop scheduling problem. Wang and Zheng [20], by combining simulated annealing and genetic algorithms, developed a general, parallel and easily implemented hybrid optimization framework, and applied it to the job shop scheduling problem. Based on an effective encoding scheme and some specific optimization operators, some benchmark job shop scheduling problems are well solved by the hybrid optimization strategy. In [17], the authors proposed a hybrid genetic algorithm. In their approach, the schedules are constructed using a priority rule in which the priorities are defined by the genetic algorithm. Schedules are constructed using a procedure that generates parameterized active schedules. After a schedule is obtained, a local search heuristic is applied to improve the solution. In [19], a hybrid method is proposed to obtain a near-optimal solution within a reasonable amount of time. This method uses a neural network approach to generate initial feasible solutions and then a simulated annealing algorithm to improve the quality and performance of the initial solutions in order to produce the optimal/near-optimal solution. Chen et al. [31] proposed an agent-based genetic algorithm that accelerates the creation of the initial population. In this approach, the processing of selection, crossover and mutation can be controlled in an intelligent way.

In this paper, we propose an agent-based parallel genetic algorithm for the job shop scheduling problem. In our approach, the initial population is created in an agent-based way by using the method proposed in [31], and then an agent-based method is used to parallelize the genetic algorithm.

The reminder of this paper is organized as follow. In Section 2, we give the details of our proposed agent-based architecture and parallel genetic algorithm. In Section 3, we discuss the implementation and experimental results of the proposed approach. Our conclusion is given in Section 4.

2. Agent-based approach for job shop scheduling problem

Agents and multi-agent systems have wide application in parallel and distributed systems. One of the important features of agents is their capability in parallel implementing of genetic algorithms. We used this feature in our approach and propose a parallel and distributed model for the job shop scheduling problem. We developed a multi-agent system containing some agents with special actions that are used to parallelize the genetic algorithm and create its population.

In this section we introduce our agent-based method and briefly describe its structure. The architecture of our method and different layers of it are introduced in Section 2.1. In Section 2.2, we describe how the initial population can be produced in an agent-based way. Our proposed method that is used to parallelize the genetic algorithm for solving the problem is explained in Section 2.3.

2.1. Agent-based architecture

The layers of our proposed agent-based architecture for the job shop scheduling problem are shown in Fig. 1. The computer network that provides a context for developing and implementation of the proposed model is at the lower layer

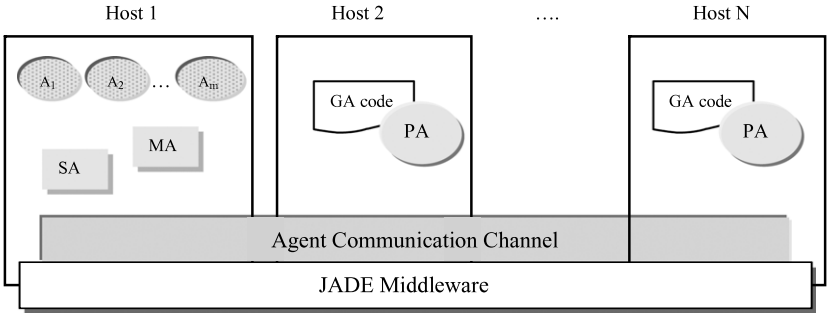

Fig. 2. Proposed agent-based architecture for the job shop scheduling problem.

of this architecture. We used JADE middleware [32] to implement our multi-agent system. JADE is a software development framework aimed at developing multi-agent systems and applications conforming to FIPA standards for intelligent agents. This middleware facilitates the creation of agents and their communication using message passing.

An overall schematic of this architecture is given in Fig. 2. Agents are distributed over various hosts in the network and JADE provides a secure communication channel for them to communicate. In this model, each agent has been developed for a special purpose. We can describe them as follows.

- *MA (Management Agent)*: *MA* and *Ai* ($i = 1, 2, \ldots, m$) agents have the responsibility of creating the initial population for the genetic algorithm. This agent controls the behaviors of *Ai*s and coordinates them in creation step.
- *Ai (Execute Agent)*: Each machine has an *Ai* agent to schedule the operations on it.
- *PA (Processor Agent)*: Each *PA* locates on a distinct host and executes the genetic algorithm on its sub-population.
- *SA (Synchronization Agent)*: This agent coordinates migration between sub-populations of *PA* agents.

Details of the agents' properties and tasks are explained in Sections 2.2 and 2.3.

2.2. Initial population

The initial population is created by using the method proposed in [31]. In this method, chromosomes of the population are produced in an agent-based way using *MA* and *Ai* ($i = 1, 2, \ldots, m$) agents. We define the job shop scheduling problem formally with the following definitions.

1. $P = \{P_1, P_2, \ldots, P_n\}$ is the set of n jobs.
2. $M = \{M_1, M_2, \ldots, M_m\}$ is the set of m machines.
3. Job P_i has k_i operations. $J_{Pi} = \{J_{Pi}(1), J_{Pi}(2), \ldots, J_{Pi}(k_i)\}$ is the set of operations of job P_i. $J_P = \{J_{P1}, J_{P2}, \ldots, J_{Pn}\}$ is the matrix of operations of n jobs. The value of $J_{Pi}(j)$ is the machine that operation j of job P_i must be processed on.
4. Let T be the $n \times m$ matrix of processing times of each job in m machines. $T[i, j]$ is the processing time of job P_i on machine j.

The status of each operation is determined by using the following definitions.

5. J_{Mi} is the set of all schedulable operations on machine M_i. A schedulable operation is an operation for which all the foregoing operations have finished. $J_M = \{J_{M1}, J_{M2}, \ldots, J_{Mm}\}$ is the set of schedulable operations on all machines.
6. *NJ* is the set of unschedulable operations, i.e. operations for which at least one of the foregoing operations has not finished.
7. *FJ* is the set of finished operations.

Two kinds of agent are used to create the initial population: the management agent (*MA*) and the execute agent *Ai* ($i = 1, 2, \ldots, m$). Each machine has an execute agent. Each *Ai* schedules the operations of its machine. The *MA* manages operations of all jobs and controls the execution of the *Ai* agents. The schedule process in Fig. 3 is used to create the initial population. Each execution of this process creates a chromosome indicating a feasible schedule for the problem instance. To create a population with size N, we execute the schedule process N times.

2.3. Parallel genetic algorithm

To parallelize our genetic algorithm we use a coarse-grain model. In this model, which is also known as an island model, the initial population is divided into sub-populations, and each sub-population is evolved separately. Communication between sub-populations is restricted to the migration of chromosomes.

In our method, after the creation of the initial population by the *MA* and *Ai* ($i = 1, 2, \ldots, m$) agents, *MA* divides it into sub-populations with the same size and sends each of them to a processor agent (*PA*). Each *PA* locates on a distinct host and executes the genetic algorithm on its sub-population independently. Different sub-populations communicate by exchanging migrants. Details of the migration policy are explained in Section 2.3.1. The parallel genetic algorithm consists of two phases: the execution phase and the migration phase. In the execution phase, the sub-populations are evolved independently by processor agents, and in the migration phase, the *PA*s exchange migrants. These two phases run repeatedly for predefined times.

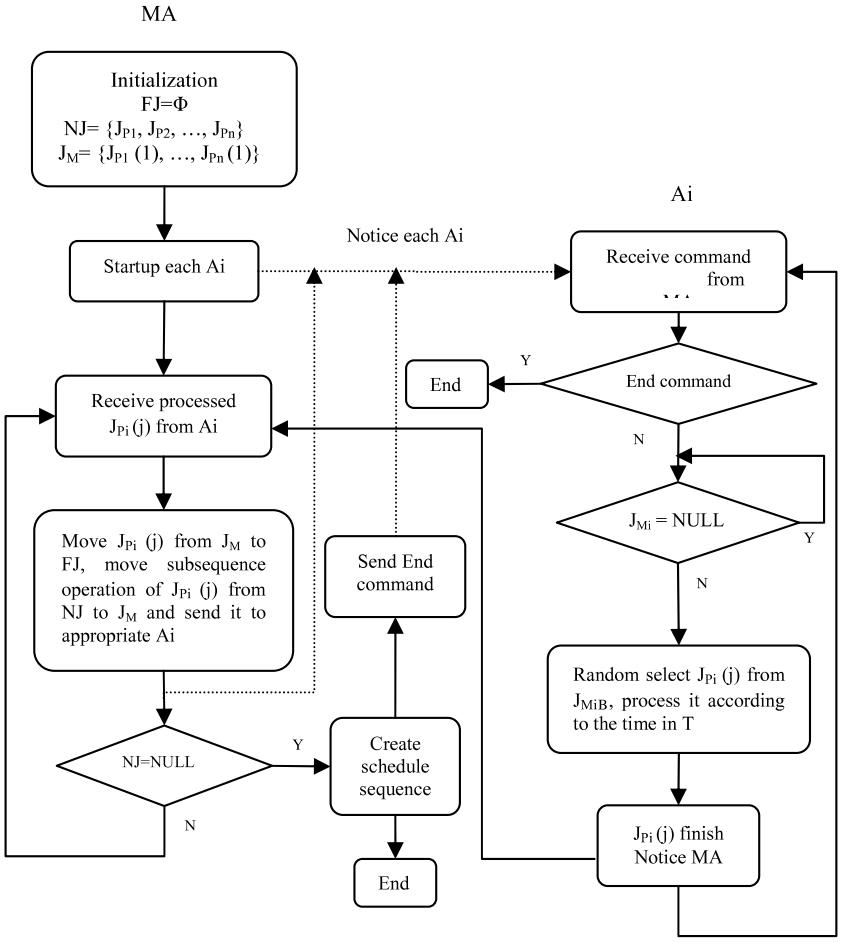

Fig. 3. Schedule process of *MA* and *Ai*.

Table 1
An example of a 4 × 4 job shop scheduling problem.

Job	Machine, processing time			
$\mathbf{P}_1$	1, 5	3, 6	2, 4	4, 3
$\mathbf{P}_2$	2, 4	1, 3	3, 11	4, 10
$\mathbf{P}_3$	3, 5	4, 8	1, 9	2, 5
$\mathbf{P}_4$	4, 10	1, 8	2, 5	3, 4

2.3.1. Migration policy

Communication between sub-populations of *PA*s is carried out by exchanging migrants. In our approach, we use a synchronous migration policy. Each *PA* executes the genetic algorithm on its sub-population for a predefined number of generations, and then it sends a message to the *SA* informing it of the end of its execution. The *SA* is a synchronization agent, which coordinates migration between sub-populations of *PA* agents. After receiving a message from all the *PA*s, the *SA* broadcasts a message to them notifying the start of migration phase. In the migration phase, each *PA* exchanges some of its best chromosomes with its neighbors. Chromosomes with low fitness value in the sub-population are replaced with the best chromosomes of neighbors. Execution of both the genetic algorithm and the migration phase is carried out repeatedly until the system converges to an acceptable solution.

2.3.2. Chromosome representation

The coding method has great importance when using genetic algorithms. We use a method that was introduced by Gen [33] to encode chromosomes of the problem. In this method, each job has a distinct number that is used to indicate its operations. In the n jobs and m machines problem, the maximum number of genes in each chromosome is $n \times m$. Job P_i appears in a chromosome k_i times. Table 1 shows an example of a 4 × 4 job shop scheduling problem. An example

Parent1: 1 4 1 3 2 1 4 2 3 1 2 4 3 2 4 3
Parent2: 4 2 3 1 2 4 3 2 4 3 1 2 1 3 4 1

Child1: 1 4 **3 1 2 4 3 2 4** 1 2 4 3 2 **1** 3
Child2: 4 2 **1 3 2 1 4 2 3** 3 1 2 1 3 4 **4**

Fig. 4. PMX example.

Before mutation: 1 4 3 1 2 **4** 3 2 4 1 2 4 3 2 1 3
After mutation: 1 4 3 1 4 **2** 3 2 4 1 2 4 3 2 1 3

Fig. 5. Mutation example.

chromosome of the problem in Table 1 is

3 2 1 4 2 3 1 2 4 3 2 4 3 1 4 1.

In the above chromosome, 1 means job P_1, 2 means job P_2, etc. Job P_1 has four operations; thus the number of times 1 appears in the chromosome is four.

One execution of the schedule process shown in Fig. 3 for this example creates such a gene sequence. To create an initial population with size N, we execute this process N times.

2.3.3. Fitness function

To determine the survival probability of a chromosome at the next generation, we need to define a fitness function. In this paper, we use a well-known fitness function proposed by Goldberg [34] to evaluate the chromosomes. This function is defined as follows:

$$Fit(C) = PT_{max} - PT(C), \tag{1}$$

where $PT(C)$ is the maximal processing time of chromosome C and PT_{max} is the maximum value of $PT(C)$. In our approach, the *MA* computes the fitness value of the chromosomes.

2.3.4. Selection

Roulette wheel selection containing the elite retaining model [34] is used to generate a new population for the next generation. In the elite retaining model, the best chromosome from the previous generation is copied to the next generation. Hence the best produced solution can never become worse from one generation to the next.

2.3.5. The crossover operator

After the selection of chromosomes, in order to create new chromosomes for the next generation a crossover operator is used. We use the partially matched crossover (*PMX*) proposed by Goldberg [35]. Two crossover points are chosen from the chromosomes randomly and equably. Then the genes of two parents that are in the area between the crossover points are exchanged (see Fig. 4). This operator can produce illegal schedules. The *MA* repairs each illegal schedule and converts it to a legal one. Details of the repair mechanism are given in Section 2.3.7.

2.3.6. The mutation operator

The mutation operator prevents the genetic algorithm from local optimums and retains population diversity. In our approach, a point from the chromosome is chosen randomly and the gene that is in this point is exchanged with its subsequent gene (see Fig. 5).

2.3.7. Repair mechanism

Creating new chromosomes by using the crossover operator may be lead to illegal schedules. A repair mechanism is used by the *MA* to convert these chromosomes to a legal form. When a *PA* creates a new chromosome, it sends it to the *MA* for repair. The *MA* replaces repeat operations of the new chromosome with absent operations to ensure that the appearance time of each job P_i is equal to k_i. The repaired chromosome is sent to the appropriate *PA*.

3. Implementation and experimental results

To evaluate the performance of our method, we carried out some experiments. We compared the solution quality of our parallel agent-based genetic algorithm with that obtained using the serial agent-based method. We used some benchmark instances for the job shop scheduling problem. These problem instances are available from the OR library web site [36]. We set the parameter values for both the serial and the parallel genetic algorithms as follows:

- population size = 1000
- generation span = 1000

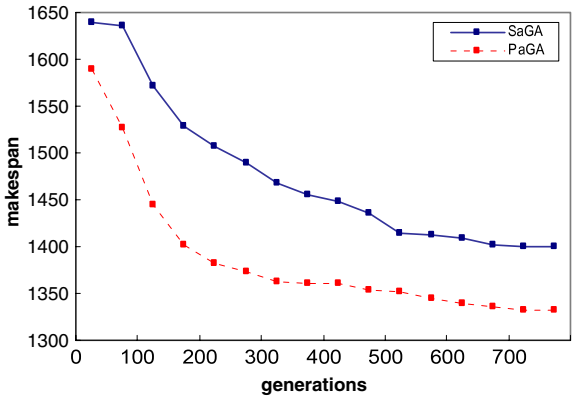

Fig. 6. PaGA finds better solutions compared with the SaGA. Test problem: LA26. Results are averages of best solutions over ten runs.

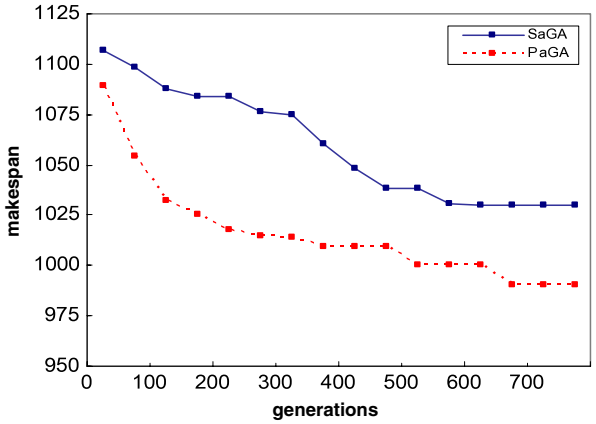

Fig. 7. PaGA finds better solutions compared with SaGA. Test problem: LA16. Results are averages of best solutions over ten runs.

- crossover rate = 0.95
- mutation rate = 0.1.

The number of *PA* agents for the parallel approach was fixed at eight in our experiments, and these agents form a virtual cube amongst themselves, and each of them has three neighbors. In the serial method we have only one *PA* agent that evolves the population that has been created by the *MA* and *Ai* ($i = 1, 2, \ldots, m$) agents.

In the parallel approach, communication between sub-populations of various *PA*s is carried out by exchanging migrants in the migration phase. The parameters of migration were set as follows:

- migration frequency: 100 generations
- migration rate: 10 chromosomes
- migration topology: cube.

These parameters indicate that each migration phase begins when the *PA*s have run the parallel genetic algorithm for 100 generations. In this phase, the ten best chromosomes from the sub-populations are exchanged between neighboring *PA*s on the cube.

To compare the parallel agent-based genetic algorithm (PaGA) with the serial agent-based genetic algorithm (SaGA) we computed the average makespan of the best schedules obtained by applying the algorithms on some problem instances during various generations. The results are shown in Figs. 6–8. These figures demonstrate the effect of applying the SaGA and PaGA on LA26, LA16 and LA32 instances. As shown by these figures, for all instances of the problem, convergence to a near-optimal solution in the parallel method takes place rapidly and the solutions that the PaGA finds in various generation numbers have shorter lengths than those that are found by the SaGA.

To evaluate how the PaGA performed with respect to solution quality we ran it on some benchmark instances. A summary of experimental results is given in Tables 2–5. The values provided in these tables show the average length of the best schedules obtained from ten executions. The columns labeled SaGA and PaGA show results based on serial and parallel approaches. The column labeled *BS* shows the best solutions obtained so far for the given benchmark instance. According to experimental results, for small sizes of the problem, the PaGA and the SaGA find solutions with similar quality, but when

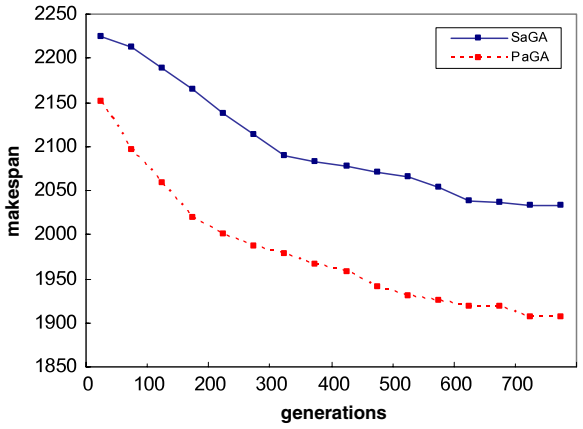

Fig. 8. PaGA finds better solutions compared with SaGA. Test problem: LA32. Results are averages of best solutions over ten runs.

Table 2
Comparisons between schedule lengths found by the PaGA and the SaGA on FT instances.

Problem	Jobs	Machines	BS	SaGA avg.	PaGA avg.	PaGA best
FT06	6	6	55	55	55	55
FT10	10	10	930	1066	1028	997
FT20	20	5	1165	1270	1224	1196

Table 3
Comparisons between schedule lengths found by the PaGA and the SaGA on ORB instances.

Problem	Jobs	Machines	BS	SaGA avg.	PaGA avg.	PaGA best
ORB01	10	10	1059	1212	1156	1149
ORB02	10	10	888	985	935	929
ORB03	10	10	1005	1190	1139	1129
ORB04	10	10	1005	1115	1065	1062
ORB05	10	10	887	976	945	936
ORB06	10	10	1010	1137	1093	1060
ORB07	10	10	397	448	422	416
ORB08	10	10	899	1048	1017	1010
ORB09	10	10	934	1039	1006	994

Table 4
Comparisons between schedule lengths found by the PaGA and the SaGA on LA instances.

Problem	Jobs	Machines	BS	SaGA avg.	PaGA avg.	PaGA best
LA01	10	5	666	666	666	666
LA02	10	5	655	690	659	655
LA03	10	5	597	633	617	617
LA04	10	5	590	618	609	607
LA05	10	5	593	593	593	593
LA06	15	5	926	926	926	926
LA07	15	5	890	890	890	890
LA08	15	5	863	863	863	863
LA09	15	5	951	951	951	951
LA10	15	5	958	958	958	958
LA11	20	5	1222	1222	1223	1222
LA12	20	5	1039	1039	1039	1039
LA13	20	5	1150	1150	1150	1150
LA14	20	5	1292	1292	1292	1292
LA15	20	5	1207	1217	1273	1207

the size of problem increases, the quality of the PaGA solutions becomes better compared with the SaGA. The PaGA not only obtains much shorter schedule lengths, but it also has higher convergence speed.

Table 5
Comparisons between schedule lengths found by the PaGA and the SaGA on LA instances.

Problem	Jobs	Machines	BS	SaGA avg.	PaGA avg.	PaGA best
LA16	10	10	945	1030	1005	994
LA17	10	10	784	835	799	793
LA18	10	10	848	926	876	860
LA19	10	10	842	896	876	873
LA20	10	10	902	955	917	912
LA21	15	10	1053	1214	1169	1146
LA22	15	10	927	1075	1023	1007
LA23	15	10	1032	1089	1051	1033
LA24	15	10	935	1065	1026	1012
LA25	15	10	977	1145	1078	1067
LA26	20	10	1218	1404	1333	1323
LA27	20	10	1269	1436	1370	1359
LA28	20	10	1216	1402	1382	1369
LA29	20	10	1195	1415	1381	1322
LA30	20	10	1355	1563	1505	1437
LA31	30	10	1784	1916	1857	1844
LA32	30	10	1850	2033	1918	1907

4. Conclusion

The job shop scheduling problem is an important and complicated problem in machine scheduling. In this paper, we have proposed an agent-based parallel genetic algorithm for this problem. Use of software agents accelerated the creation of the initial population of genetic algorithm. To enhance the performance of our genetic algorithm, we parallelized it using an agent-based method. We compared the performance of our parallel agent-based approach with that of a serial agent-based method. The results showed that the parallel method improves the quality of the solutions obtained. Future work will concentrate on improving the performance of our method and applying it to similar problems.

References

[1] A.S. Jain, S. Meeran, Deterministic job-shop scheduling: past, present and future, Department of Applied Physics and Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, UK, 1998.
[2] J. Carlier, E. Pinson, An algorithm for solving the job shop problem, Management Science 35 (29) (1989) 164–176.
[3] B.J. Lageweg, J.K. Lenstra, A.H.G. Rinnooy Kan, Job shop scheduling by implicit enumeration, Management Science 24 (1977) 441–450.
[4] P. Brucker, B. Jurisch, B. Sievers, A branch and bound algorithm for job-shop scheduling problem, Discrete Applied Mathematics 49 (1994) 105–127.
[5] V.R. Kannan, S. Ghosh, Evaluation of the interaction between dispatching rules and truncation procedures in job-shop scheduling, International Journal of Production Research 31 (1993) 1637–1654.
[6] R. Vancheeswaran, M.A. Townsend, A two-stage heuristic procedure for scheduling job shops, Journal of Manufacturing Systems 12 (1993) 315–325.
[7] Z. He, T. Yang, D.E. Deal, Multiple-pass heuristic rule for job scheduling with due dates, International Journal of Production Research 31 (1993) 2677–2692.
[8] J. Adams, E. Balas, D. Zawack, The shifting bottleneck procedure for job shop scheduling, Management Science 34 (1988) 391–401.
[9] E. Nowicki, C. Smutnicki, A fast taboo search algorithm for the job-shop problem, Management Science 42 (6) (1996) 797–813.
[10] S.G. Ponnambalam, P. Aravindan, S.V. Rajesh, A tabu search algorithm for job shop scheduling, International Journal of Advanced Manufacturing Technology 16 (2000) 765–771.
[11] P.V. Laarhoven, E. Aarts, J.K. Lenstra, Job shop scheduling by simulated annealing, Operations Research 40 (1992) 113–125.
[12] J.B. Chambers, Classical and flexible job shop scheduling by tabu search, Ph.D. Dissertation, University of Texas at Austin, Austin, TX, 1996.
[13] M.E. Aydin, T.C. Fogarty, Simulated annealing with evolutionary processes in job shop scheduling, in: Evolutionary Methods for Design, Optimisation and Control, (Proceeding Of EUROGEN 2001, Athens, 19–21 Sept.) CIMNE, Barcelona, 2002.
[14] M. Kolonko, Some new results on simulated annealing applied to job shop scheduling problem, European Journal of Operational Research 113 (1999) 123–136.
[15] T. Satake, K. Morikawa, K. Takahashi, N. Nakamura, Simulated annealing approach for minimizing the makespan of the general job-shop, International Journal of Production Economics 60–61 (1999) 515–522.
[16] F.D. Croce, R. Tadei, G. Volta, A genetic algorithm for the job shop problem, Computers and Operations Research 22 (1995) 15–24.
[17] J.F. Goncalves, J.J.d.M. Mendes, M.G.C. Resende, A hybrid genetic algorithm for the job shop scheduling problem, European Journal of Operational Research 167 (2005) 77–95.
[18] L. Wang, D.Z. Zheng, A modified genetic algorithm for job shop scheduling, International Journal of Advanced Manufacturing Technology (2002) 72–76.
[19] R.T. Mogaddam, F. Jolai, F. Vaziri, P.K. Ahmed, A. Azaron, A hybrid method for solving stochastic job shop scheduling problems, Applied Mathematics and Computation 170 (2005) 185–206.
[20] L. Wang, D.Z. Zheng, An effective hybrid optimization strategy for job-shop scheduling problems, Computers & Operations Research 28 (2001) 585–596.
[21] S.Y. Foo, Y. Takefuji, H. Szu, Scaling properties of neural networks for job shop scheduling, Neurocomputing 8 (1) (1995) 79–91.
[22] J. Zhang, X. Hu, X. Tan, J.H. Zhong, Q. Huang, Implementation of an ant colony optimization technique for job shop scheduling problem, Transactions of the Institute of Measurement and Control 28 (2006) 93–108.
[23] K.L. Huang, C.J. Liao, Ant colony optimization combined with taboo search for the job shop scheduling problem, Computers & Operations Research 35 (2008) 1030–1046.
[24] J. Montgomery, C. Fayad, S. Petrovic, Solution representation for job shop scheduling problems in ant colony optimization, Faculty of Information & Communication Technologies, Swinburne University of Technology, 2006.
[25] M. Ventresca, B. Ombuki, Ant colony optimization for job shop scheduling problem, in: Proceeding of 8th IASTED International Conference on Artificial Intelligence and Soft Computing, ASC 2004, CDROM:451-152, 2004.

[26] S. Petrovic, C. Fayad, A genetic algorithm for job shop scheduling with load balancing, School of Computer Science and Information Technology, University of Nottingham, Nottingham, 2005.
[27] S. Rajakumar, V.P. Arunachalam, V. Selladurai, Workflow balancing in parallel machine scheduling with precedence constraints using genetic algorithm, Journal of Manufacturing Technology Management 17 (2006) 239–254.
[28] B.M. Ombuki, M. Ventresca, Local search genetic algorithms for the job shop scheduling problem, Applied Intelligence 21 (2004) 99–109.
[29] S.C. Lin, E.D. Goodman, W.F. Punch, Investigating parallel genetic algorithms on job shop scheduling problems, Genetic Algorithm Research and Applications Group, State University of Michigan, Michigan, 1995.
[30] R. Cheng, M. Gen, Y. Tsujimura, A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: Hybrid genetic search strategies, Computers & Industrial Engineering 36 (1999) 343–364.
[31] Y. Chen, Z.Z. Li, Z.W. Wang, Multi-agent-based genetic algorithm for JSSP, in: Proceedings of the Third International Conference on Machine Learning and Cybernetics, 2004, pp. 267–270.
[32] F. Bellifemine, A. Poggi, G. Rimassa, Developing multi-agent systems with a FIPA-compliant agent framework, Software: Practice and Experience 31 (2001) 103–128.
[33] M. Gen, Y. Tsujimura, Genetic algorithms for solving multiprocessor scheduling problems, in: Proceeding of 1st Asia–Pacific Conference on Simulated Evolution and Learning, 1997, pp. 106–115.
[34] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, MA, 1989.
[35] D.E. Goldberg, R. Lingle, Alleles, loci, and the TSP, in: Proceeding of 1st International Conference on Genetic Algorithms, 1985, pp. 154–159.
[36] D.C. Mattfeld, R.J.M. Vaessens, Job shop scheduling benchmarks, Available: OR Library online, http://mscmga.ms.ic.ac.uk (accessed 10.07.08).