IEMS Vol. 10, No. 1, pp. 7-14, March 2011.

A Shaking Optimization Algorithm for Solving Job Shop Scheduling Problem

Ehab A. Abdelhafiez[†]
Mechanical and Industrial Engineering Department
Faculty of Engineering, Majmaah University, Majmaah, Saudi Arabia
Tel: +966-53-000-5671, E-mail: ehabaty@yahoo.com

Fahd A. Alturki
Electrical Engineering Department
Faculty of Engineering, King Saud University, Riyadh, Saudi Arabia
Tel: +966-53-000-5671, E-mail: falturki@ksu.edu.sa

Received, January 17, 2011; Revised, February 14, 2011; Accepted, February 21, 2011

Abstract. In solving the Job Shop Scheduling Problem, the best solution rarely is completely random; it follows one or more rules (heuristics). The Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Simulated Annealing, and Tabu search, which belong to the Evolutionary Computations Algorithms (ECs), are not efficient enough in solving this problem as they neglect all conventional heuristics and hence they need to be hybridized with different heuristics. In this paper a new algorithm titled "Shaking Optimization Algorithm" is proposed that follows the common methodology of the Evolutionary Computations while utilizing different heuristics during the evolution process of the solution. The results show that the proposed algorithm outperforms the GA, PSO, SA, and TS algorithms, while being a good competitor to some other hybridized techniques in solving a selected number of benchmark Job Shop Scheduling problems.

Keywords: Job Shop Scheduling Problem, Evolutionary Computation, Optimization, Genetic Algorithm(GA), Intelligent Systems, Shaking Optimization Algorithm(SOA)

1. INTRODUCTION

The Job Shop Scheduling Problem (JSSP) may be roughly sketched as follows: we are given a set of jobs and a set of machines. Each job needs a number of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine. Each machine can handle at most one job at a time. The purpose is to find a schedule that is an allocation of the operations to time intervals on the machines that has minimum length. According to Peter Brucker (2007), examples of practical applications of JSSP are problems in flexible manufacturing, multiprocessor task scheduling, robotic cell scheduling, railway scheduling, and air traffic control which all have an underlying job-shop structure. The JSSP is considered as a particularly hard combinatorial optimization problem. Only very special cases of the problem can be solved in polynomial time, but their immediate generalizations are NP-hard.

Since the mid-sixties, many general-purpose optimization algorithms have been proposed for finding near-optimal solutions to numerical, real-valued 'black-box' problems for which exact and analytical methods do not apply; most notably: Evolutionary Programming (EP), Evolution Strategies (ES), and Genetic Algorithms (GA). Recently, Particle Swarm Optimization (PSO) and Differential Evolution (DE) have been introduced. And finally Electromagnetism Algorithm (EM) has been proposed as explained by Liya *et al.* (2007). Such algorithms are belongs to the Evolutionary Computations Algorithms (EC). They are based on the processes of evolution in nature. They simulate the evolution of a population of solutions based on the measured performance (fitness) of each member of the population. Simulated evolution is implemented by the application of various operators, such as mutation and recombination between different solutions, as well as through selection rules used to determine which members of the population survive and reproduce. With each subsequent generation, the population is likely to show improvement in

† : Corresponding Author

overall performance, until some terminating condition is met (Anthony, 2002).

These optimization algorithms showed some type of limitations as they neglect all conventional heuristics. In most of the NP-hard problems, the best solution rarely be completely random, it follows one or more rules (heuristics). For example, in one-machine job shop scheduling problem, the best solution can be found using the Shortest Processing Time (SPT) rule. While, in two-machine job shop scheduling problem, the best solution can be found using Johnson Heuristic. In solving cutting stock problems, Abdelhafiez (2008) has studied 81 different ordering rules He found that the decreasing area is among the group of rules that give efficient results with all problem sets, while the increasing area rule are among the rules that give poor results.

Recently, and because of the limitations associated with the evolutionary computations algorithms, it becomes a common practice to improve the performance of these tools by incorporating different search and heuristic techniques. As an example, in solving JSSP, Moraglio *et al.* (2007) stated that, because of the complementary properties of genetic algorithms and conventional heuristics, the hybrid approach often outperforms either method operating alone. Accordingly, they proposed a genetic local search algorithm (GTS) consisting of a basic genetic algorithm with the addition of a Tabu search optimization phase. Kamrul Hasan *et al.* (2009), after analyzing the traditional GA solutions during solving JSSPs, they realized that the solutions could be further improved by applying simple rules or local search. Consequently, they improved the performance of their proposed GA by incorporating a simple priority rules. Fatih Tasgetiren *et al.* (2006) stated that both PSO and DE algorithms need to be hybridized with an efficient local search method based on a variable neighborhood search (VNS) method in order to improve the solution quality. They stated also that the major obstacle of successfully applying PSO and DE algorithms to combinatorial optimization problems is due to their continuous nature and to remedy this drawback, the smallest position value (SPV) rule is employed.

In general, the JSSP has been studied by many authors and several algorithms have been proposed. The recent research on JSSP is focused on Evolutionary Computations Algorithms such as Simulated Annealing (SA) (Steinhofel *et al.*, 1999, Emin Aydin and Terence, 2004), Taboo Search (TS) (Senthil and Selladurai 2007, Zhang *et al.*, 2008), Variable Neighborhood Search (Mladenovic and Hansen, 1997, Mehmet and Emin, 2006), Genetic Algorithm (GA) (Dirk and Christian 2004, José *et al.*, 2005), Ant Colony Optimization (ACO) (Ventresca and Ombuki, 2004), Particle Swarm Optimization (PSO) (Fatih, 2006), and Neural Network (NN) (Jain and Meeran, 1998). A comprehensive survey of the JSS problem can be found in (Albert and Luis, 1998, Chandrasekharan and Oliver, 1999).

In this paper, a new optimization algorithm titled "Shaking Optimization Algorithm (SOA)" is proposed to be used to solve the JSSP. The algorithm emulates shaking a box that was filled with different objects in order to nest these objects in the minimum required space or to align them correctly to give a room to add more ones.

This paper presents the structure of the proposed algorithm, the main procedures that are to be followed in applying it, and the values of its parameters. At the end, a comparative study is presented with the GA, PSO, SA and TS, through solving a selected number of benchmark instances of the JSSP.

2. OVERVIEW OF THE SHAKING OPTIMIZATION ALGORITHM

The proposed algorithm follows the common methodology of the Evolutionary Computations. In EC, simulated evolution is implemented by the application of various operators such as mutation and recombination in a random manner. The proposed algorithm is a structured search algorithm that applies a number of heuristics during this search process. It starts with reordering the operations (of all jobs) according to some criteria (weight). Due to this rearrangement, gaps may exist. To close these gaps, another heuristic can be applied. The process of reordering and gap closing is to be repeated many times until some stopping criteria is met.

It is a hierarchy of four consecutive stages: local search, collision, fine tuning and global optimization.

2.1 Local Search

The proposed approach assumes that the solution at any time is a set of objects of different sizes that are ordered in one raw tangent to each other. The objective is to find the order that gives the minimum length of that raw. This can be done through shaking these objects in all directions (x&-x, y&-y, z&-z) searching for the shortest possible raw.

The coordinate system is not a real x-y-z system but it is a system of priority rules that manage the searching process of the solutions. So, one can assume that shaking along the x-axis, as an example, force the objects to reorder themselves in an ascending order according to their size as a result of their inertia, and so on.

2.2 Collision

During the searching process, if a gap exists between object (A) and object (B) then, due to the shaking process object (A) will hit (B) giving it a push in the same direction of the shake forcing it to jump to another location. If this new location is not better than the original one, then object (B) will return back resulting in a push to (A), which in return will jump to another location in the direction of this push.

2.3 Fine Tuning

At the end of each shake or number of repeated shakes, the movement of the objects will start to slow-down resulting in swapping of some objects where gaps exist around them. In addition, some objects may rotate around its center resulting in better penetration, according to the nature of the problem under consideration, as in the case of cutting and packing problem where items may be allowed to rotate as shown by E. Abdelhafiz *et al.* (2001).

2.4 Global Optimization

To recover from or avoid penetration in a local optimum, then after a certain number of iterations, or at any other criteria, the current solution is to be kept in solution-list, and then a strong shake is to be applied to generate a new initial solution. This new solution will include segments (blocks) from the last one. At the end of the solution process, the best solution out of the solution-list is to be considered as the final solution.

3. PROCEDURES OF THE SHAKING ALGORITHM

The proposed algorithm utilizes a number of heuristic rules through the four consecutive stages as follows:

3.1 Stage 1, Local Search

The problem is to be represented using a suitable coding system, and then a random initial solution is to be developed. Following this, the coordinate system that will be used in the evolution of this random solution is to be identified. This coordinate system is to be represented by a selected number of priority rules. The shaking force is to be defined after, where this force defines the number of elements (objects) that are going to change their positions at each shake. Each of the selected rules is to be applied individually, then if improvement exists, the resulting solution will be kept, and the remaining rules should be applied on this new solution.

To apply priority rules in this stage, it is required first to assign a weight for each element (object). This weight may be: time, area, penalty, relative importance ⋯ etc. according to the nature of the problem under consideration. The rules proposed to represent different coordinates are:

- Ascending-order, *according to the selected criteria (weight), represents movement along x-axis direction*
- Descending-order, *according to the selected criteria, represents movement along-x-axes direction,*

The above two rules emulate the action of inertia as a result of the shaking process where items with heavy weights will try to move towards the end of the sequence.

- Concave-order, *according to the selected criteria, represents movement along y-axes direction*,
- Convex-order, *according to the selected criteria, represents movement along-y-axes direction,*

The above two rules are corresponding to the effect of inertia as well. Items with heavy weights will try to move towards y-direction (the center of the order) or-y direction (the terminals of the order), according to the direction of movement, making the sequence to be similar to a wire that stretched from middle where it's both ends are fixed.

- Meshing of one small-item among large-ones*, corresponding to movement along Z-direction,*
- Meshing of more than one small-item among large-ones*, corresponding to movement along Z-direction*,

The last rules, represent the results that can be obtained when one shakes some items up and down where small items will try to penetrate (nest) among large ones.

3.2 Stage 2, Collision:

The objects that may hit each other are those objects have gap in-between. The definition of "gap" depends on the problem under consideration. The gap may be the delay time of a job or be the idle time of a machine as in the Job Shop Scheduling problem, where it may be the difference in the number of entities or be the difference between areas as in the Cutting Stock Problem ⋯ etc.

At the end of each shake, a scanning process is to be carried out to find gap(s) between objects, and once a gap is found, the insertion heuristic rule is to be applied that execute the collision process.

3.3 Stage 3, Fine Tuning:

Slowing down the movement of the objects allows some items to swap or rotate that may result in improving the solution. The items allowed to swap are those items come just after a gap or followed by a gap, i.e., those items at the beginning or at the end of a block. The rule that most describing this swapping action is the one of Nowicki and Smutnicicki (1996) and has been described by Emin and Terence (2004) as shown in Figure 1.

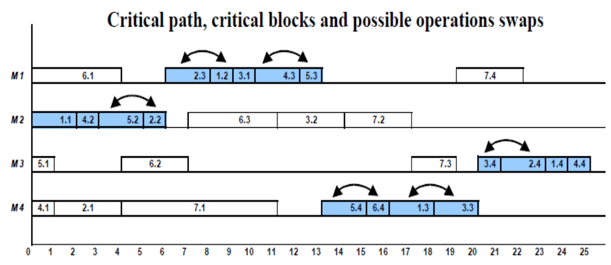

Figure 1. Neighborhood of Nowicki and Smutnicicki (1996).

3.4 Stage 4, Global optimization:

The shake to be applied to generate a new initial solution from the last one will keep some blocks from the current solution but may change their positions randomly. The blocks that will remain are the compact blocks which have no gaps. So, although this shake will generate new solution with less efficiency than the original one, it guarantees that the chance for this new solution to evolve in a converging manner in high.

A Pseudo-code for the Shaking Optimization Algorithm (SOA) is shown in Figure 2.

4. PARAMETERS OF THE SHAKING OPTIMIZATION ALGORITHM

For any optimization algorithm, it is very important to set the values of the parameters of that algorithm properly in order to guarantee a good flow during the evolutionary process in order to obtain best performance of the algorithm. The parameters regarding the SOA algorithm, that were found affect the quality of the algorithm performance, are:

- Population size (number of solutions),
- Number of iterations for each solution,
- Number of dimensions,
- Weight criteria,
- Shaking force,

5. APPLICATION OF THE SOA TO SOLVE THE JOB SHOP SCHEDULING PROBLEM

5.1 Problem Representation

The problem representation described by Mehmet and Emin (2006) is used here. Schedules are represented

```
•   Generate initial solution, Sol1
•   While stopping criteria for the number of Solutions not satisfied, repeat:
    {
    •   Considering current solution,
    •   While stopping criteria for the number of iterations for each solution not satisfied, repeat:
        {
        (Shaking):
            ➢   For number of dimensions selected, repeat:
            •   Develop random number RND
            •   Starting from item # RND to item # (RND + shake force), Do:
                •   Reorder items in the range according to the priority rule associated with current dimension
                •   Evaluate current position,
                •   Keep position if improvement exists
        (Collision):
            ➢   Scan for Gap existence between different items, then:
            ▪   Apply insertion priority rule, evaluate current position, and keep position if improvement exists,
        (Fine Tuning):
            ➢   Scan for Gap existence between different items, then:
            ▪   Apply swapping priority rule, evaluate current position, and keep position if improvement exists,
        }
    •   Copy current solution to Sol_{t+1}
    •   Move current solution to Solution-List, then
    •   Considering new solution Sol_{t+1}, :
        {
            ➢   Scan for Blocks of items where no gap exists within any block
            ➢   Apply strong Shake considering each Block as one item while changing positions
        }
    }
```

Figure 2. Pseudo-code for the Shaking Optimization Algorithm (SOA).

using a set of integers; each integer stands for an operation. It is also called chromosome of (n×m) gene representing a problem of n-jobs, m-machines. Each job is represented m-times within the chromosome, while these integers do not represent certain operations. This way of representation prevents infeasibility, and always provides a feasible active schedule. As an example, if there is a chromosome of [2 1 2 2 1 3 1 3 3], where {1, 2, 3} represents {job$_1$, job$_2$, job$_3$} respectively. Obviously, there are totally 9 operations, but, 3 different integers, each is repeated 3 times. The integer on the first gene, 2, represents the first operation of the second job. Likewise, the integer on the second gene, 1, represents the first operation of the first job on corresponding machine. Thus, the chromosome of [2 1 2 2 1 3 1 3 3] is understood as [o_{21}, o_{11}, o_{22}, o_{23}, o_{12}, o_{31}, o_{13}, o_{32}, o_{33}] where o_{ij} stands for the i_{th} operation of j_{th} job.

5.2 SOA Parameters

A great attention has been paid in finding the best setting of the proposed SOA algorithm in case of solving JSSP. The setting of the algorithm parameters for the JSSP is as follows:

- The population size = n×m (n-jobs, m-machines),
- Number of iterations = 10×n×m,
- Number of dimensions = 6,
- Weight criteria is processing time, and
- Shaking force = random number between 3 and 7.

The algorithm will terminate if a pre specifies result is achieved (e.g., if the best known solution is known) without the need to complete all number of iterations.

5.3 Computational Results

In order to evaluate the contribution of the proposed SOA algorithm in solving the JSSP compared with the performance of most common evolutionary computation algorithms: Genetic Algorithm (GA), Particle Swarm Optimization algorithm (PSO), Simulated Annealing algorithm (SA), and Tabu Search algorithm (TS), computational experiments were carried out for solving 14 selected benchmark instances of JSSP test problems of different sizes given by Lawrence (1984) and found at OR-Library (http://mscmga.ms.ic.ac.uk).

The proposed SOA algorithm is compared with the following algorithms as found in literature:

Genetic Algorithms:

- Ventresca and Ombuki (2003)
- Kamrul Hasan *el al.* (2009)

Particle Swarm Optimization

- Pisut Pongchairerks and Voratas (2007)

Simulated Annealing

- Ventresca and Ombuki (2003)

Tabu Search

- Ventresca and Ombuki (2003)

Table 1 summarizes the results. It lists instance name, problem size (n×m), the best known solution (BKS), and the solution obtained by each algorithm.

The results in table 1 above show that the Shaking Optimization Algorithm is a good competitor for the evolutionary computation algorithms under study (GA, PSO, SA, and TS) in solving all of the selected benchmark instances. It gives the best known solutions for 13 instances out of the 14 selected ones; La01, La02, La03,

Table 1. Experimental results, SOA VS. GA, PSO, SA, and TS.

Instance	Size	BKS	GA (2003)	GA (2009)	PSO (2007)	SA (2003)	TS (2003)	SOA
La01	10×5	**666**	667	667	**666**	**666**	**666**	**666**
La02	10×5	**655**	676	**655**	704	659	**655**	**655**
La03	10×5	**597**	627	617	630	620	617	**597**
La06	15×5	**926**	**926**	**926**	**926**	**926**	**926**	**926**
La07	15×5	**890**	891	**890**	922	**890**	**890**	**890**
La08	15×5	**863**	**863**	**863**	884	**863**	**863**	**863**
La11	20×5	**1222**	**1222**	**1222**	**1222**	**1222**	**1222**	**1222**
La12	20×5	**1039**	**1039**	**1039**	**1039**	**1039**	**1039**	**1039**
La13	20×5	**1150**	**1150**	**1150**	**1150**	**1150**	**1150**	**1150**
La16	10×10	**945**	993	994	1047	974	986	**945**
La17	10×10	**784**	804	794	865	796	796	**784**
La18	10×10	**848**	874	861	888	871	866	**848**
La19	10×10	**842**	895	890	958	871	870	**842**
La20	10×10	**902**	942	976	995	957	941	907

La06, La07, La08, La11, La12, La13, La16, La17, La18, and La19 while, none of the other algorithms do. In addition, for instances; La03, La16, La17, La18, and La19 the proposed algorithm is the only one that gives the BKS. And regarding the last instances La20, the proposed algorithm SOA gives near BKS and still outperforms all of the other algorithms.

In addition to the above comparison with most common evolutionary computation algorithms: GA, PSO, SA, and TS, another comparison was carried out between the proposed SOA algorithm and a selected number of hybridized algorithms. The hybridized algorithms selected are:

HGA:

A Hybrid Genetic Algorithm by Goncalves *et al.* as stated by Emin and Terence (2004).

The algorithm combines a genetic algorithm, a schedule generator procedure that generates parameterized active schedules, and a local search procedure which in turn employ two exchange local search based on the disjunctive graph model of Roy and Sussmann (1964) and the neighborhood of Nowicki and Smutnicicki (1996),

PSO_{VNS}:

Particle Swarm Optimization hybridized with an efficient local search method based on a Variable Neighborhood Search method (VNS) by Fatih *et al.* (2006),

SAGA:

Hybrid Simulated Annealing with Genetic Algorithm by Ventresca and Ombuki (2003),

TS:

First-last neighborhood structure with dynamic Tabu length by Senthil Vemurugan and Selladurai (2007).

Table 2 summarizes the results of the comparison.

The results in Table 2 above show that the Shaking Optimization Algorithm is still good competitor for the hybridized evolutionary computation algorithms in solving all the selected benchmark instances. It outperform the SAGA by Ventresca and Ombuki (2003), and it gives the same results exactly as do the Hybrid Genetic Algorithm by Goncalves *et al.* (2004), while the Tabu search by Senthil Vemurugan and Selladurai (2007) outperform the two algorithms regarding the last instance La20, but fail to give the same result for instance La16. In addition, the proposed algorithm outperforms all other algorithms regarding solution times. As an example, the solution times for instances La01 to La10 using the HGA algorithm range from 37 seconds for La01 to 99 seconds for La08 while, these times range from 2 seconds to 60 seconds using the proposed algorithm.

6. CONCLUSIONS

In this paper a new algorithm titled "Shaking Optimization Algorithm (SOA)" is proposed to be used in solving the Job Shop Scheduling Problem (JSSP). The proposed algorithm follows the common methodology of the Evolutionary Computations where, evolution is implemented by the application of various operators. This SOA algorithm is a structured search algorithm which has low degree of randomness compared with most of other Evolutionary Computations Algorithms. The algorithm emulates the actual shaking process. It consists of four stages; namely, local search, collision, fine tuning and global optimization. It starts with reordering the operations according to some criteria and due to this rearrangement, gaps may exist. To close these gaps, heuristic rules can be applied. The process of reor-

Table 2. Experimental results, SOA vs. HGA, PSOVNS, SAGA, and TS.

Instance	Size	BKS	HGA (2004)	PSO_{VNS} (2006)	SAGA (2003)	TS (2007)	SOA
La01	10×5	**666**	**666**	-	-	**666**	**666**
La02	10×5	**655**	**655**	-	-	**655**	**655**
La03	10×5	**597**	**597**	-	-	**597**	**597**
La06	15×5	**926**	**926**	-	-	**926**	**926**
La07	15×5	**890**	**890**	-	-	**890**	**890**
La08	15×5	**863**	**863**	-	-	**863**	**863**
La11	20×5	**1222**	**1222**	-	-	**1222**	**1222**
La12	20×5	**1039**	**1039**	-	-	**1039**	**1039**
La13	20×5	**1150**	**1150**	-	-	**1150**	**1150**
La16	10×10	**945**	**945**	**945**	988	946	**945**
La17	10×10	**784**	**784**	-	793	**784**	**784**
La18	10×10	**848**	**848**	-	862	**848**	**848**
La19	10×10	**842**	**842**	**842**	874	**842**	**842**
La20	10×10	**902**	**907**	-	926	**902**	907

dering and gap closing is to be repeated many times until some stopping criteria is met. A selected 14 instances of benchmark problems from different sizes has been solved, and the results obtained show that the proposed algorithm outperforms GA, PSO, ST, and TS. In addition, the results show that, the SOA algorithm outperforms the Genetic Algorithm hybridized with Simulated Annealing Algorithm, while it gives the same results exactly as does the Genetic Algorithm hybridized with two exchange local search.

The main contribution of the proposed algorithm is its effectiveness with respect to implementation and processing time as it works alone without the need for hybridization with other algorithms.

REFERENCES

Moraglio, A., Ten Eikelder, H. M. M., and Tadei, R. (2001), Genetic local search for job shop scheduling, *Technical Report CSM-435 ISSN*, 1744-8050.

Jain, A. S. and Meeran, S. (1998), Job shop scheduling using neural networks, *International Journal of Production Research*, **36**, 1249-1272.

Albert Jones, and Luis C. Rabelo (1998), Survey of job shop scheduling techniques, *NISTIR*, *National Institute of Standards and Technology*, *Gaithers-burg*, *MD*.

Anthony Jack Carlisle (2002), Applying the particle swarm optimizer to non-stationary environments, *A Dissertation Submitted to the Graduate Faculty of Auburn University in Partial Fulfillment of the Requirements of the Degree of Doctor of Philosophy*, *Auburn University*.

Chandrasekharan Rajendran, and Oliver Holthaus (1999), A comparative study of dispatching rules in dynamic flow shops and job shops, *European Journal of Operational Research*, **116**, 156-170.

ChaoYong Zhang, PeiGen Li,YunQing Rao, and ZaiLin Guan (2008), A very fast TS/SA algorithm for the job shop scheduling problem, *Computers and Operations Research*, **35**, 282-294.

Dirk C. Mattfeld a, and Christian Bierwirth (2004), An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European Journal of Operational Research*, **155**, 616-630.

Abdelhafiez, E. (2008), Efficient items-ordering rules in cutting stock problem, *9th International Conference on Mechanical Design and Production Engineering MDP9, Cairo-Egypt*.

Abdelhafiz, E., Elmaghraby, A. S., and Hassan, M. F. (2001), A genetic approach for 2-D irregular cutting stock problems, *Proceedings of ISCA 10th international Conference on intelligent Systems*, *USA*, 114-117.

José Fernando Gonçalves, Jorge José de Magalhães Mendes, and Maurício G. C. Resende (2005), A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, **167**, 77-95.

Steinhofel, K., Albrecht, A. and Wong, C. K. (1999), Two simulated annealing-based heuristics for the job shop scheduling problem, *European Journal of Operational Research*, **118**, 524-548.

Liya GU, Alexandru Sava, Sophie Hennequin, and Xiaolan XIE (2007), Electromagnetism-like mechanism algorithm for stochastic assembly line balancing with reliability constant, *Proceeding of International Conference on Industrial Engineering and System Management*, *IESM*, *Beijing-China*.

Emin Aydin, M. and Terence C. Fogarty (2004), A simulated annealing algorithm for multi-agent systems: a job shop scheduling application, *Journal of Intelligent Manufacturing*, **15**.

Fatih Tasgetiren, M., Mehmet Sevkli, Yun-Chia Liang, and Mutlu Yenisey, M. (2006), A particle swarm optimization and differential evolution algorithms for job shop scheduling problem, *International Journal of Operations Research*, **3**, 120-135.

Ventresca, M. and Ombuki, B. M. (2003), Meta-heuristics for the job shop scheduling problem, *Technical Report # CS-03-12.*

Ventresca, M. and Ombuki, B. M. (2004), Ant Colony optimization for job shop scheduling problem, *Technical Report # CS-04-04.*

Mehmet Sevkli, and Emin Aydin, M. (2006), Collaborating variable neighbourhood search algorithms for job shop scheduling problems, *Proceedings of 5th International Symposium on Intelligent Manufacturing Systems,* 450-461.

Mehmet Sevkli, and Emin Aydin, M. (2006), Variable Neighbourhood Search for job shop scheduling problems, *Journal Of Software*, **1**, 34-39.

Mladenovic, N. and Hansen, P. (1997), Variable Neighborhood Search. *Computers and Operations Research*, **24**, 1097-1100.

Senthil Vemurugan, P. and Selladurai, V. (2007), A Tabu Search Algorithm for job shop scheduling problem with industrial scheduling case study, *International Journal Of Soft Computing*, **2**(4), 531-537.

Peter Brucker (2007), The job-shop problem: old and new challenges. MISTA.

Pisut Pongchairerks, and Voratas Kachitvichyanukul (2007), A comparison between algorithms VNS with PSO and VNS without PSO for job-shop scheduling problems. *International Journal of Computational Science*, **1**, 179-191.

Kamrul Hasan, S. M., Ruhul Sarker, Daryl Essam, and

David Cornforth (2009), Memetic Algorithms for solving job-shop scheduling problems, *Memetic Computing Journal*, **1**, 69-83.

Vesterstrom, J. and Thomsen, R. (2004), A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, *Evolutionary Computation*, *CEC2004*, *Congress on*, **2**(19-23), 1980-1987.