Variable Neighbourhood Search for Job Shop Scheduling Problems

Mehmet Sevkli
Operation Research Department, Defense Sciences Institute,
Turkish Military Academy
06654, Cankaya, Ankara
Email: msevkli@kho.edu.tr

M. Emin Aydin
University of Bedfordshire, Dept. of Computing and Information Systems
Luton, UK
Email: mehmetaydin@acm.org

***Abstract*— Variable Neighbourhood Search (VNS) is one of the most recent metaheuristics used for problem solving in which a systematic change of neighbourhood within a local search is carried out. In this paper, an investigation on implementing VNS for job shop scheduling problems is carried out tackling benchmark suites collected from OR library. The idea is to build the best local search and shake operations based on neighbourhood structure available. The results are presented and compared with the recent approaches in the literature. It is concluded that the VNS algorithm can generally find better results.**

Index Terms—**variable neighbourhood search, job shop scheduling**

I. INTRODUCTION

Metaheuristics are general strategies for designing heuristic procedures to solve optimization problems via searching through the solution space. The heuristic search procedures offer moves around the neighbourhood of a particular solution in some particular fashions, which are called neighbourhood structures. Variable neighbourhood search (VNS) is a recent metaheuristic for solving combinatorial and global optimization problems whose basic idea is systematic change of neighbourhood within a local search. It is based on a simple principle: change the neighbourhood structure when the search is trapped on a local minimum, which very likely happens in most of combinatorial and/or multi-model numerical optimization problems. Especially, as search space grows fast with enlarging problem sizes, the likelihood of being trapped in local minima becomes inevitable. The main interest of research in this field is to make clear the way of recovering trapped search or to put effort for keeping on-course. VNS offers a multiple neighbourhood structure with which one recovers the solutions trapped via the others. The main idea here is to choose heuristics (neighbourhood structures) complementary to each other.

Job Shop Scheduling (JSS) problems with the objective function of minimizing makespan (C_{max}) is one of the best known and strongly NP-hard [1] combinatorial problems. Among the benchmarks within the literature, small size instances can be solved within reasonable computational time by exact algorithms such as branch-and-bound approach [2, 3]. However, when the problem size increases, the computational time of the exact methods grows exponentially. Therefore, the recent research on JSS problems is focused on heuristic algorithms such as Simulated Annealing (SA) [4, 5, 6], Genetic Algorithm (GA) [7, 8, 9, 10], Taboo Search (TS) [11, 12, 13], Ant Colony Optimization (ACO) [14, 15], Neural Network (NN) [16], Shifting Bottleneck Procedure [17, 18], Guided Local Search [19], Parallel Greedy Randomized Adaptive Search Procedure (GRASP) [20] and Constraint Propagation [21]. A comprehensive survey of the JSS problem can be found in [22].

In this paper, we examine a further investigation of VNS is carried out to make clear the contribution which has been published preliminary by us [23]. For this purpose, we have examined 7 VNS algorithm provided with their functional configurations.

The organization of this paper is as follows. Section II introduces the background information of JSS, the way how to represent the problems and the neighbourhood structure employed. The third section is about VNS algorithm implemented. The extensively carried out experiments are reported and discussed in the fourth section while the fifth section provides with the conclusions.

II. BACKGROUND

The background information provided in the following subsections is mainly about job shop scheduling, problem representation and the alteration structure for search.

A. Problem Description

Job Shop Scheduling (JSS) problems have been studied for a long time. Due to its NP-Hard nature, this type of problems has never been dropped from scientific

research interests and kept becoming a popular testbed for metaheuristics.

The problem is comprised of a set of jobs (J) to be processed on a set of machines (M) subject to a number of technological constraints. Each job consists of m operations, $O_j=\{o_{1j},...,o_{mj}\}$, each operation must be processed on a particular machine, and there is only one operation of each job to be processed on each machine. There is a predefined order of the operations of each particular job in which each operation has to be processed after its predecessor (PJ_j) and before its successor (SJ_j). In the end of the whole schedule, each machine completes processing n operations in an order that is determined during the scheduling time, although there is no such order initially. Therefore, each operation processed on machine M_i has a predecessor (PM_i) and a successor (SM_i). A machine can process only one operation at a time. There are no set-up times, no release dates and no due dates.

Each operation has a processing time (p_{ij}) on related machine starting at the time of r_{ij} . The completion time of operation o_{ij} is therefore: $c_{ij}=r_{ij}+p_{ij}$, where $i = (1,...,m)$, $j = (1,..., n)$ and $r_{ij} = \max(c_{iPJ_j}, c_{PM_{i,j}})$. Machines and jobs have particular completion times, which are denoted and identified as: $C_{M_i} = c_{in}$ and $C_{J_j} = c_{in}$ where c_{in} and c_{jm} are the completion time of the last (n^{th}) operation on i^{th} machine and the completion time of the last (m^{th}) operation of j^{th} job, respectively. The overall objective is to minimize the completion time of the whole schedule (makespan), which is the maximum of machines' completion times, $C_{max} = max(C_{M1}, ..., C_{Mm})$.

B. Problem Representation

Schedules are represented in a set of integers, where each stands for an operation. It is also called chromosome of $n \times m$ gene representing a problem of n jobs, m machines. Since each integer does not represent a certain operation, but the last completed operation of corresponding job, each job is represented m times within the chromosome. This way of representation prevents infeasibility, and always provide with a feasible active schedule. For instance, we are given a chromosome of [2 1 2 2 1 3 1 3 3], where {1, 2, 3} represents {j_1, j_2, j_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 to be processed first on corresponding machine. 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. More details can be found in [24].

C. Neighbourhood Structure

The neighbourhood structure with which the neighbouring solutions are determined to move to is one of the key elements of metaheuristics, as the performance of the metaheuristic algorithm significantly depends on the efficiency of the neighbourhood structure. The following two neighbourhood structures are employed in this study:

- ***Exchange*** is a function used to move around in which any two randomly selected operations are simply swapped. For instance, suppose that we are given a state of [2 1 2 2 1 3 1 3 3] and the two random numbers derived are 2 and 8. After applying *Exchange*, the new state will be [2 1 3 2 1 3 1 3 2]. Obviously, the 2nd and 8th genes of the chromosome were 2 and 3, respectively. Applying *Exchange* function, the new 2nd and 8th genes were swapped and turned to 3 and 2, respectively.
- ***Insert*** is another fine-tuning function that inserts a randomly chosen gene in front or back of another randomly chosen gene. For instance, we are given the same state as before. In order to apply *Insert*, we also need to derive two random numbers; one is for determining the gene to be inserted and the other is for the gene that insertion to be done in front/back of it. Let us say those number are 3 and 6, where 3rd gene is 2 and the 6th one is 3. Consecutively, the new state will be [2 1 2 1 2 3 1 3 3].

Although there are many other, maybe more efficient, neighbourhood structures reported in the literature, we preferred these two due to the simplicity and ease of use alongside a reasonable efficiency. The others such as critical path based-functions, provide more efficiency, but definitely require much more computational time and experience and hard working.

III. VARIABLE NEIGHBOURHOOD SEARCH

Variable neighbourhood search (VNS) is one of the most recent metaheuristics developed for problem solving in an easier way. It is known as one of very well-known local search methods [25], gets more attention day-by-day, because of its ease of use and accomplishments in problem solving. Basically, a local search algorithm carries out exploration within a limited region of the whole search space. That facilitates a provision of finding better solutions without going further investigation. The VNS is a simple and effective search procedure that proceeds to a systematic change of neighbourhood. An ordinary VNS algorithm gets an initial solution, $x \in S$, where S is the whole set of search space, than manipulates it through a two nested loop in which the core one alters and explores via two main functions so called *shake* and *local search*. The outer loop works as a diversifier reiterating the inner loop, while the inner loop carries the major search. *Local search* explores for an improved solution within the local neighbourhood, whilst *shake* diversifies the solution by switching to another local neighbourhood. The inner loop iterates as long as it keeps improving the solutions, where an integer, k, controls the length of the loop. Once an inner loop is completed, the outer loop re-iterates until the termination condition is met. Since the complementariness of neighbourhood functions is the key idea behind VNS, the neighbourhood structure / heuristic functions should be

chosen very rigorously so as to achieve an efficient algorithm.

In order to develop an effective VNS algorithm, one needs two kinds of neighbourhood functions, $N_k^s(x)$ and $N_l^{LS}(x)$ resulting each with a particular neighbourhood structure, where $N_k^s(x)$ and $N_l^{LS}(x)$ denote neighbourhood functions for *shake* and *local search*, respectively. The neighbourhood structures used may be more than one for each function (*shake* and *local search*) so as to achieve a valuable neighbourhood change. For that purpose, the indices, *k* and *l*, are to be used for *shake* and *local search* functions, respectively, in order to ease switching from one to another neighbourhood. Obviously, both indices have upper boundaries, which are denoted with k_{max} and l_{max}. Hence, $1 \le k \le k_{\max}$ and $1 \le l \le l_{\max}$ are the ranges identified for each indices.

The VNS comprises the following steps:

1. **Initialization**: Find an initial solution *x*.
2. **Repeat** the following steps until the stopping condition is met:
   (a) **Shake Procedure:** Generate at random a starting solution $x' \in N_k^s(x)$.
   (b) **Local Search:** Apply a local search from the starting solution *x'* using the base neighborhood structure $N_l^{LS}(x)$ until a local minimum $x'' \in N_l^{LS}(x)$ is found.
   (c) **Improve or not:** If *x"* is better than *x*, do $x \leftarrow x'$.

If the local search uses greedy strategy, then at Step 2(b) an iterative procedure tests all the base moves providing the best neighbouring solution until a local minimum is obtained. The shake procedure selects randomly a solution from $N_l^{LS}(x)$.

In this paper, we developed the *local search* function as a simple variable neighbourhood descent algorithm based on both aforementioned neighbourhood structures. It keeps iterating as long as better moves achieved. It stops, if the algorithm produces a predefined number of non-better moves consecutively. Once the local search finishes a run, then the *shake* function works to switch to another region so as to carry out a new *local search* there, as *shake* functions to diversify the exploration. The *local search* procedure is provided as follows.

1. **Get** initial solution, $x \in S$
2. **Set** $k \leftarrow 1$
3. **while** $k \le k_{max}$ **do**
   if $(k=1)$ **then** $x'' \in S \leftarrow$ Exchange(x')
   else if $(k=2)$ **then** $x'' \in S \leftarrow$ Insert(x')
   if $f(x'') < f(x')$ **then** $x' \leftarrow x''$ and $k \leftarrow 1$
   else $k \leftarrow k+1$

On the other hand, the *shake* function consists of a couple of repeated random moves conducted by the neighbourhood function described above The shake procedure has set to operating with exchange and insert functions successively. Given state *x** is operated with *Exchange* to obtain *x'*, which is operated then with *Insert* consecutively. Finally, Exchange re-operates on the outcome of Insert, say *x"*, to obtain *x*.

IV. Experimental Study

In this paper, we provided experimental results for various VNS algorithms in order to clarify the efficiency of implementations. The measures considered in this study are mainly about quality of solution and/or computational time. The success of the algorithm regarding the quality of solution has mainly been accounted with respect to the relative percentage of error (RPE) index, which is calculated as follows:

$$RPE = \frac{(bf - opt)}{opt} \times 100 \quad (1)$$

where *bf* is the best makespan found and *opt* is either the optimum or the lowest boundary known for uncertain optimum values. Obviously, RPE is calculated based on the best value found, and also it can be measured benchmark-by-benchmark. In order to review the results in a broader point of view, we developed a second index based on the latter RPE calculation averaged over the 30 repetitions. That is called ARPE standing for averaged relative percentage of error. The third index used is the hitting-ratio (HR) being calculated as the number of optimum found through the whole repetitions. This is needed as other indexes may not build a sufficient level of confidence with the results. The whole software was coded in C and run on an Intel Pentium IV 2.6 GHz PC with 256MB memory. The JSS benchmark suits, which are very well known within the field, were picked up from OR-Library [26].

A. Experimentation with VNS algorithms

We have examined a number of VNS implementations; each differs from the other with the configuration of shake and local search functions by using the neighbourhood structures. The idea is to develop efficient implementation. Following are the list of 7 VNS algorithms provided with their functional configurations.

VNS-I	:	Shake ←Exchange +Insert + Exchange, Local Search ←Exchange +Insert
VNS-II	:	Shake← Exchange +Insert + Exchange, Local Search ← Insert + Exchange
VNS-III	:	Shake← Exchange +Insert+ Exchange, Local Search ← Exchange
VNS-IV	:	Shake← Exchange+ Insert+ Exchange, Local Search ← Insert
VNS-V	:	Shake← Insert+ Insert+ Insert, Local Search ← Insert
VNS-VI	:	Shake← Exchange+Exchange+ Exchange, Local Search ← Exchange
VNS-VII	:	No Shake, Local Search ← Exchange + Insert

In Table 1, the experimental results obtained by each VNS algorithms with respect to the quality of solutions measured in ARPE and HR indexes, where former (APRE) is minimized and the latter (HR) is maximized. The experiments have been conducted over 31 benchmarks tackled; some are known moderately hard but some are very hard. The accomplishments of the algorithms are clearly reflected. The APRE and HR indexes provide very consistently, which proves that measuring the performances reflects the real achievement. The table comprises of 7 main parts, where each part consists of two columns; one shows the achievement with respect to APRE and the other does with HR. The first two columns are allocated for the name and optimum values of each benchmark while the rest are devoted to each version of VNS. Obviously, VNS-I and VNS-II are competitive for the best while VNS-VII provides the worst as its HR values remain mostly around 0% and the mean of APRE is the highest. On the other hand, VNS-II, which seems the best, provides with APRE of 0.76 % and HR of 34 %. The other 4 versions remain competitive with one another. The main aspect shared by the first two versions is their local search algorithms, which comprises of two neighbourhood structures (NS) while the other four versions those have the intermediate performance made of a local search of a single NS. The shake function does not create so much change, apparently, as the shake function of VNS-III and VNS-IV are the same as VNS-I and VNS-II while the shake of VNS-V and VNS-VI vary. As a result, we observe the significant change with a local search of double NS versions. The version with the worst performance, VNS-VII, does not have a proper shake function, though a double NS local search operates.

TABLE I. RESULTS OBTAINED FROM SEVEN VNS ALGORITHMS WITH RESPECT TO 2 INDEXES FOR QUALITY OF SOLUTIONS

Benchmark		VNS-I		VNS-II		VNS-III		VNS-IV		VNS-V		VNS-VI		VNS-VII	
Prob.	Opt.	ARPE (%)	HR	ARPE (%)	HR	ARPE (%)	HR	ARPE (%)	HR	ARPE (%)	HR	ARPE (%)	HR	ARPE (%)	HR
ft10	930	0.55	0.60	0.94	0.47	1.11	0.40	1.17	0.33	1.02	0.40	1.87	0.10	4.79	0.00
ft20	1165	0.54	0.50	0.46	0.53	0.95	0.10	0.77	0.30	0.87	0.20	0.90	0.17	1.08	0.03
la16	945	0.30	0.50	0.57	0.50	0.46	0.40	0.34	0.57	0.83	0.30	0.24	0.57	3.04	0.07
la19	842	0.09	0.93	0.09	0.93	0.15	0.87	0.19	0.80	0.13	0.87	0.29	0.67	2.66	0.03
abz05	1234	0.10	0.60	0.11	0.60	0.22	0.40	0.20	0.40	0.22	0.30	0.15	0.60	1.43	0.20
abz06	943	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.01	0.97	1.53	0.13
orb01	1059	1.47	0.13	1.58	0.00	2.62	0.03	1.70	0.03	1.95	0.00	1.93	0.00	5.21	0.00
orb02	888	0.33	0.00	0.33	0.00	0.38	0.00	0.38	0.00	0.42	0.00	0.36	0.00	2.60	0.00
orb03	1005	2.60	0.10	1.76	0.37	2.72	0.10	3.20	0.10	3.43	0.10	2.64	0.10	7.52	0.00
orb04	1005	0.62	0.40	0.50	0.40	0.93	0.13	0.72	0.30	0.84	0.13	0.92	0.10	2.69	0.00
orb05	887	0.31	0.17	0.62	0.07	0.76	0.00	0.66	0.03	1.08	0.00	0.74	0.03	4.07	0.00
orb06	1010	0.89	0.10	0.78	0.03	1.21	0.00	0.96	0.00	1.00	0.03	1.21	0.00	5.91	0.00
orb07	397	0.27	0.80	0.25	0.80	0.48	0.63	0.28	0.80	0.58	0.57	0.52	0.57	2.49	0.17
orb08	899	1.63	0.23	0.83	0.50	1.28	0.50	1.74	0.33	1.98	0.30	1.93	0.30	5.58	0.03
orb09	934	0.78	0.13	0.65	0.30	0.74	0.17	0.60	0.30	0.62	0.23	0.72	0.20	2.34	0.00
orb10	944	0.00	1.00	0.00	1.00	0.08	0.87	0.04	0.93	0.04	0.93	0.04	0.93	4.86	0.07
abz07	656	2.01	0.00	2.09	0.00	2.70	0.00	2.35	0.00	2.12	0.00	2.95	0.00	3.11	0.00
abz08	665	1.99	0.00	1.83	0.00	2.86	0.00	2.19	0.00	2.39	0.00	2.82	0.00	3.76	0.00
abz09	679	2.17	0.00	2.16	0.00	2.88	0.00	2.42	0.00	2.61	0.03	2.81	0.00	3.65	0.00
la21	1046	0.62	0.03	0.77	0.00	1.30	0.00	0.86	0.00	0.94	0.03	1.09	0.00	2.74	0.00
la22	927	0.24	0.57	0.18	0.63	0.67	0.13	0.34	0.47	0.42	0.30	0.64	0.13	1.28	0.03
la24	935	0.60	0.03	0.63	0.00	1.29	0.00	0.71	0.03	0.88	0.03	1.32	0.00	2.74	0.00
la25	977	0.64	0.07	0.49	0.23	0.99	0.00	0.81	0.07	0.74	0.17	1.07	0.00	2.27	0.03
la27	1235	0.91	0.00	1.00	0.03	1.80	0.00	1.38	0.03	1.42	0.03	1.70	0.00	1.65	0.03
la28	1216	0.03	0.87	0.03	0.90	0.42	0.17	0.12	0.67	0.13	0.67	0.36	0.23	0.30	0.53
la29	1152	1.80	0.00	1.92	0.00	3.04	0.00	2.42	0.00	2.17	0.00	3.13	0.00	3.40	0.00
la36	1268	0.49	0.27	0.48	0.37	1.07	0.00	0.75	0.07	0.86	0.17	1.00	0.03	1.98	0.00
la37	1397	0.73	0.37	0.83	0.30	0.98	0.20	0.92	0.13	1.03	0.07	1.14	0.17	2.41	0.07
la38	1196	0.97	0.07	0.82	0.07	1.85	0.00	1.37	0.03	1.78	0.07	1.38	0.00	4.19	0.00
la39	1233	0.43	0.30	0.37	0.40	1.07	0.00	0.61	0.07	0.60	0.20	1.01	0.00	2.58	0.03
la40	1222	0.42	0.00	0.42	0.00	0.84	0.00	0.69	0.00	0.64	0.00	0.74	0.00	2.25	0.00
Average		**0.79**	**0.32**	**0.76**	**0.34**	**1.22**	**0.20**	**1.00**	**0.25**	**1.09**	**0.23**	**1.22**	**0.19**	**3.10**	**0.05**

A. Related Works

The related works has been discussed earlier in introduction section. The theme of this subsection is to bring forward a comparison between our results and the results gained by works related to job shop scheduling recently published in order to build a level of confidence. Table 2 presents results provided with various metaheuristics recently published and VNS-II configuration with respect to the quality of the solutions in PRE index. The reason to switch to RPE index back is due to the difficulty of calculating APRE index with the results provided in the related literature. The benchmarks chosen are those which considered very hard among the list of 31 in Table 1. These algorithms taken into account are listed as follows:

- Distributed evolutionary simulated annealing algorithm (dESA) by Aydin and Fogarty [4].
- Ant colony optimization algorithm (ACO GSS) by Blum and Sampels [14].

- Parallel GRASP with path-relinking (GRASP) by Aiex et al.[20]
- A Hybrid Genetic Algorithm (HGA) by Goncalves et al.[10].
- A Tabu Search Method (TSSB) by Pezzella and Merelli [13].

Since the best of our VNS algorithms in Table 1 is VNS-II, we put the results obtained with that algorithm. We can observe that the VNS-II outperform the other algorithms compared with respect to RPE index, as the lowest value provided by TSSB is about 0.88 while VNS-II provides with 0.24. On the other hand, VNS-II algorithms remain competitive among themselves, as one provides better for one benchmark but worse for another. VNS-II provides definitely better than 4 algorithms.

TABLE II. A COMPARISON AMONG THE META-HEURISTICS RECENTLY PUBLISHED WITH RESPECT TO THE QUALITY OF THE SOLUTION IN RPE INDEX.

Benchmarks		**Related works**					
Problems	**Optimum**	**dESA [4]**	**ACO GSS [14]**	**GRASP [20]**	**HGA [10]**	**TSSB [13]**	**VNS-II**
abz07	656	2.44	2.74	5.49	NA	1.52	0.46
abz08	665	2.41	3.61	6.02	NA	1.95	0.60
abz09	679	2.95	3.39	8.98	NA	2.06	0.15
la21	1046	0.00	0.10	1.05	0.00	0.00	0.00
la24	935	0.32	0.96	2.03	1.93	0.32	0.00
la25	977	0.00	0.00	0.72	0.92	0.20	0.00
la27	1235	0.40	0.65	2.75	1.70	0.00	0.08
la29	1152	2.08	1.39	4.43	3.82	1.39	0.95
la38	1196	0.42	2.59	1.84	1.92	0.42	0.00
la40	1222	0.49	0.49	1.80	1.55	0.90	0.16
Average		**1.15**	**1.59**	**3.51**	**1.69**	**0.88**	**0.24**

V. CONCLUSIONS

In this paper, we examined VNS algorithms for job shop scheduling problems, which has been studied for far long time. Because of its hardness and being representative for planning problems, many methods have been tested with this family of problems. In this paper, the VNS implementations have been tested with respect to the efficiency over classical job shop problems. The best of the implementations has been exploited in investigation of VNS algorithms. It has been shown that the VNS implementation has done well and outperformed a number of meta-heuristics recently published.

REFERENCES

[1] Garey, M. Johnson, D., and Sethy, R. :The Complexity of Flow Shop and Job Shop Scheduling. *Mathematics of Operations Research* 1 (1976) 117–129.

[2] Applegate, D., and Cook, W. :A Computational Study of Job-Shop Scheduling. *ORSA Journal on Computing* 3(2) (1991) 149–156

[3] Carlier, J., and Pison, E. :An Algorithm for Solving the Job-Shop Problem, *Management Science* 35 (1989) 164–176.

[4] Aydin, M. E., and Fogarty, T. C.: A Distributed Evolutionary Simulated Annealing Algorithm for Combinatorial Optimisation Problems. *Journal of Heuristics* 10 (2004) 269–292.

[5] Kolonko, M. :Some New Results on Simulated Annealing Applied to the Job Shop Scheduling Problem. *European Journal of Operational Research* 113, (1999) 123–136.

[6] Satake, T., Morikawa, K., Takahashi, K., and Nakamura, N. :Simulated Annealing Approach for Minimizing the Makespan of the General Job- Shop. *International Journal of Production Economics* 60 (1999) 515–522.

[7] Bierwith, C. :A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms. *OR Spektrum* 17 (1995) 87–92.

[8] Dorndorf, U., and Pesch, E.: Evolution Based Learning in a Job Shop Scheduling Environment, *Computers & Operations Research* 22 (1995)

[9] Groce, F. D. Tadei, R., and Volta, G. :A Genetic Algorithm for the Job Shop Problem. *Computers & Operations Research* 22 (1995) 15–24.

[10] Goncalves, J. F., Mendes, J. M., and Resende, M. :A hybrid genetic algorithm for the job shop scheduling problem, *European Journal of Operations Research* 167(1) (2004) 77-95.

[11] Nowicki, E., and Smutnicki, C. :A Fast Taboo Search Algorithm for the Job Shop Problem. *Management Science* 42 (1996) 797–813.

[12] Dell'Amico, M., and Trubian, M. :Applying Tabu-Search to the Job-Shop Scheduling Problem. *Annals of Operations Research* 4 (1993) 231–252.

[13] Pezzella, F. and Merelli, E. :A Tabu Search Method Guided by Shifting Bottleneck for the Job Shop Scheduling Problem. European *Journal of Operational Research* 120:297–310, 2000.

[14] Blum, C., and Sampels, M. :An Ant Colony Optimization Algorithm for Shop Scheduling Problems. *Journal of Mathematical Modelling and Algorithms* 3 (2004) 285–308.

[15] Colorni, A., Dorigo, M., Maniezzo, V., and Trubian, M. :Ant System for Job-Shop Scheduling. *Belgian Journal of Operations Research*, Statistics and Computer Science (JORBEL) 34(1) (1994) 39-53.

[16] Satake, T., Morikawa, K., Takahashi, K., and Nakamura, N. :Neural Network Approach for Minimizing the Makespan of the General Job- Shop. *International Journal of Production Economics* 33 (1994) 67–74.

[17] Adams, J., Balas, E., and Zawack, D.: The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science* 34 (1988) 391–401.

[18] Huang, W., and Yin, A. : An Improved Shifting Bottleneck Procedure for the Job Shop Scheduling Problem. *Computers & Operations Research* 31 (2004) 2093–2110.
[19] Balas, E., and Vazacopoulos, A. :Guided Local Search with Shifting Bottleneck for Job Shop Scheduling. *Management Science* 44 (1998) 262-275.
[20] Aiex, R. M., Binato, S., and Resende, M. G. C.: Parallel GRASP with Path-Relinking for Job Shop Scheduling. *Parallel Computing* 29 (2003) 393–430.
[21] Dorndorf, U., Pesch, E., and Phan-Huy, T.: Constraint Propagation and Problem Decomposition: A Preprocessing Procedure for the Job Shop Problem, *Annals of Operations Research* 115 (2002) 125–145.
[22] Jain, A., and Meeran, S. : Deterministic Job-Shop Scheduling: Past, Present and Future. *European Journal of Operational Research.* 113: (1999) 390–434.
[23] Sevkli, M., and Aydin, M. E. : A variable neighbourhood search algorithm for job shop scheduling problems, *Lecture Notes in Computer Science* 3906, (2006), 261-271.
[24] Cheng, R. , Gen, M., and Tsujimura, Y. :A Tutorial Survey of Job Shop Scheduling Problems Using genetic Algorithms-I. Representation. *Journal of Computers and Industrial Engineering* 30(4) (1996) 983-997.
[25] Mladenovic, N., and Hansen, P. :Variable Neighborhood Search. *Computers and Operations Research* 24 (1997) 1097–1100
[26] Beasley, J.E. "Obtaining Test Problems via Internet." *Journal of Global Optimisation* 8, 429-433, http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

Mehmet Sevkli is an Assistant Professor in the Defense Sciences Institute of Turkish Military Academy, Ankara, Turkey. He received his B.S. in Industrial Engineering from Sakarya University, his M.Sc. in Industrial Engineering from Fatih University and his Ph.D. in Industrial Engineering, from Istanbul Technical University.

His research interests include Meta-heuristics algorithms applied to the combinatorial optimization problem such as Scheduling problems.

Mehmet Emin Aydin is a research/teaching fellow in Department of Computing and Information Systems of the University of Bedfordshire, UK. He received his B.Sc. in Industrial Engineering from Istanbul Technical University, his M.A. in Production Management from Istanbul University and his Ph.D. in Intelligent Manufacturing Systems from Sakarya University.

His research interests include combinatorial optimization, meta-heuristics, evolutionary computation, intelligent agents and multi agent systems, grid computing and network planning and optimization. He has recently conducted leading guest editorial for International Journal of Production Research for the special issue on Advances in Evolutionary Computation for Manufacturing and Design Problems. Besides being member of advisory committee of many international conferences, he keeps membership of editorial board of European Journal of Industrial Engineering (EJIE). He is currently a member of The OR Society UK, ACM and IEEE Computer Society.