Heuristics for Job Shop Scheduling with Volatile Machine Constraints

Oliver Krauss, Daniel Wilfing and Andreas Schuler
University of Applied Sciences Upper Austria
Hagenberg, Austria 4232
Email: {oliver.krauss}, {daniel.wilfing}, {andreas.schuler} @fh-hagenberg.at

***Abstract*—An alteration of the job shop scheduling problem, concerning advertisement scheduling on digital advertisement spaces, is presented. Dispatching Rules (DR), Iterated Local Search (ILS) and Genetic Algorithms (GA) are discussed and applied to the problem space. The results show that ILS is the best performing heuristic, and surpasses the other heuristics especially in large problem spaces ($\geq$ 100 machines, $\geq$ 100 jobs). The results match previously made findings, which indicates that effects on large-scale problems should be further researched in conjunction with amalgam algorithms between DR, GA and ILS.**

***Keywords*—job shop, genetic algorithms, local search, distribution rules, scheduling**

I. Introduction

The job shop problem (JSP) is a well known optimization problem and a number of different heuristics have been researched as a solution to this problem. While job shop scheduling has many variations, such as waiting times on machines, or jobs needing to be executed on specific machines, the problem always requires a multitude of jobs to be scheduled on one or more machines. The schedule must be optimized towards one or more objective, such as finding a schedule of minimal length, or minimal tardiness on finishing jobs. [1], [3]

The JSP presented in this article has some unusual constraints which stem from the problem domain, which is the scheduling of advertisement jobs to electronic displays. This results in some unique differences from regular JSP.

An advertisement job distinguishes itself from a regular factory as follows:

Unique operation: Unlike a regular factory job, which usually consists of several different steps that need to be performed to create a product [4], a job in the context of this article has only one single operation that is repeated several times. This operation is playing the advertisement on one electronic display.

Parallelism: While producing one physical product can rarely be parallelized on several different machines, every advertisement job may be processed on any number of machines at the same time.

Schedulable times: This is a unique constraint that would not be considered in regular JSPs. The time of a day a job is allowed to be played (adult content, etc.) restricts the times a job can occur in a schedule.

Constraints on job ordering: A job may never be run on the same machine directly before or after a job that is in direct competition (same branch) to it. This can be compared to job shop scheduling, that requires set-up times between different jobs, but is not completely identical to this.

Flexible machine selection: Similar to flexible JSP (FJSP), where a job can be processed on any machine on a given set of machines[3], a job can also be processed on any machine positioned in an advertisement region, such as a state or city.

Machines that process the advertisement jobs have unique constraints as well:

Stochastic, volatile standby time: In a factory machine the standby time of the machine is known beforehand [1]. This is not the case in advertising where the standby time is defined by the length of time people are in close proximity to the advertisement space. Since this length of time is unknown beforehand, the problem at hand is stochastic. An approximation of the standby time can be reached by collecting statistics on past standby times. However, while analysing the problem, it has been observed that these times are highly volatile.

Throughput The length of an advertisement is known in advance. This is similar to the fixed length of an operation in JSP [1]. However, a playback has different qualities depending on the desired target audience that shall be reached. One person of the target audience seeing the advertisement is called an impression, and was used as the measurement for throughput. Thus the operation time to finish the job actually varies depending on the target audience that can be reached on a machine, meaning one machine may be able to finish a job faster than another one.

This paper shows approaches how to solve the version of JSP at hand and compares their applicability on the problem domain. Methods (2) shows the algorithms that were applied to the problem domain. Results (3) shows a comparison between these algorithms. Finally these results are discussed (4) and an outlook on further possible research (5) is given.

II. Methods

The methods for this problem were conducted as an offline scheduling approach as opposed to online. The difference is that in online scheduling the schedules are calculated during machine operation and can thus react to immediate changes in the environment, while offline scheduling requires the entire schedule to be calculated beforehand [13]. Online Scheduling could not be applied because the distributed machines do not have a constant contact to the scheduling environment, and thus aren't able to request further scheduling instructions.

Usually a schedule is simply a list of instructions that are to be executed in order. In this context however, every instruction is connected to a data-file (e.g. video, image) that needs to be played. Due to limitations in processing power and storage capabilities, it was further necessary to ensure that a schedule for a machine is restricted in size and complexity. As a representation form of the JSP, a job based representation [cheng:survey1] was selected. For each machine a list of jobs are to be executed in the form of a round-robin schedule.

Several heuristics were tested for applicability to the problem space. All Heuristics have two performance measures to be optimized towards. The first performance measure is the amount of generated impressions of the desired target audience of the job divided by the target audience available at the machine inferred through statistical analysis (equation 1). The second performance measure is minimizing the tardiness of an advertisement job (equation 2). Both measures are equally important and thus, when determining quality, are added together as can be seen in equation 3.

$dtaj = DesiredTargetAudienceforJob$
$tta = TotalTargetAudience$
$gi = GeneratedImpressions$
$rij = RemainingImpressionsforJob$
$rd = RemainingDays$

$$p1 = \sum_{n=1}^{Machines} (\frac{dtaj_n}{tta_n}) * gi \tag{1}$$

$$p2 = \left| \frac{rij}{rd} - \sum_{n=1}^{Machines} gi \right| \tag{2}$$

$$QualityforSchedule = \sum_{n=1}^{Jobs} (p1_n + p2_n) \tag{3}$$

A. Dispatching Rules

They are also called priority rules and are a greedy construction heuristic, that are used to determine which job is processed next on a machine. Every time a machine runs out of instructions and requires new ones the dispatching rule selects the job with the highest priority from a pool of available jobs. The priorities of the jobs are determined by one or more performance measurements, such as satisfying job-due-dates, or minimizing the make-span of jobs. [6], [5]

Dispatching rules are often used in online scheduling due to the fact that the rules calculate a schedule incrementally, meaning that even partially calculated schedules can be used instantly since they won't be changed anymore. However they can also be applied offline by calculating the entire schedule beforehand. [5], [14]

In recent years efforts have been made to improve, or completely autogenerate priority rules using genetic programming [3], [15].

The projects were prioritized by their remaining impressions divided by the amount of days the projects were still schedulable (equation 4). Due to the offline scheduling nature the machines were not selected when they need more instructions, but rather pre-sorted by the impressions they can produce for the desired target audience and then selected in order (equation 5).

$$PriorityJobs = \frac{rij}{rd} \tag{4}$$

$$PriorityMachines = (\frac{dtaj_n}{tta_n}) * gi \tag{5}$$

B. Iterated Local Search

Local Search (LS) is a methaheuristic that continually improves a solution by applying a small change to the solution and accepting it, if the change improves the overall solution quality, which is determined by one or more performance criteria. LS can get stuck in a local minimum.

Iterated Local Search (ILS) is an improvement upon LS. ILS applies LS iteratively after finding a local optimum and restarting the search with a modified version thereof, until either the global optimum is reached or a stop condition applies. The modification in ILS is often called a kick, since it is applied to kick the LS out of a local optimum by mutating a bigger part of the solution. When kicking, it is necessary to find an appropriate kick size, since too small kicks result in staying in the local optimum while bigger kicks result in a loss of information on the LS. [8], [10], [9]

Tabu Search, an alternative method of applying LS, is another promising option of applying search to JSP. It is hard to apply on huge problem sizes (the JSP in question needs to be able generate schedules for $\geq$ 1000 machines and jobs) due to keeping a tabu list of already visited solutions and the resulting growth of memory-usage. Thus Tabu Search was not available to comparison in the problem area. [8], [11]

The application of the shifting bottleneck heuristic [12] was not applied in the problem since all machines are capable of playing any job and jobs are independent from each other, thus no bottleneck will occur. Instead the LS of the ILS selects a random machine to be optimized and randomly switches out one job.

The ILS that was used is using a Random LS as local search step. The LS cycles through every machine and randomly mutates the schedule of one machine and applies the solution if the mutation results in a better global quality, or ignores the mutation if the quality becomes worse or stays the same. The cycling is usually not done in ILS, but rather a number of random mutations is applied [9]. However this solution makes the LS independent of the problem size itself.

As an alternative to mutating the entire schedule of one machine in the local search step a random mutation of one entry in the schedule was also thought of. This proved to be a too small mutation early on in testing (little to no quality improvement during LS, even during runs with ten thousand repeats), and was thus not further researched.

The kick of the implemented ILS mutates a percentage of the solution by randomly generating new schedules for some of the machines.

C. Genetic Algorithm

Genetic Algorithms (GA) have successfully been applied over a multitude of JSP instances. However, due to their high demand in processing time they are exclusively used in offline scheduling. [13]

GA apply the process of natural selection to solution instances of a problem, such as JSP. In GA a single solution is thought of as a chromsome that consists of a gene sequence. This gene sequence can be manipulated (mutated) or crossed with other chromosomes. First an initial population of valid solutions is generated, usually randomly. Each of the solutions is then evaluated according to a fitness function. Afterwards, the next generations will be generated based on the previous generation. Until the full next generation is built, two or more solutions are selected by a selection operator (tournament, stochastic, ...), similar to survival of the fittest, and then crossed with each other (one-point, two-point, uniform, ...), similar to a breeding process. Each new solution may be mutated, where one or more genes are randomly changed in the solution, similar to the biological evolutionary process. Research suggests that the mutation rate should be low (between 0.25% and 2%) and is correlated to the selection pressure of the GA [7]. There is also an option to apply elitism where a few of the the best solutions of the previous generation will be carried over to the next [17]. [16], [2]

Recent attempts have been made to hybridize GA with other machine learning algorithms. [19] mixes GA with dispatching rules with promising results. In the publication chromosomes are generated with the basic GA algorithm, including crossover and mutation, but instead of the chromosomes being solutions, the solutions are generated with dispatching rules. In another application [18] GA is mixed with fuzzy selection that replaces the usual fitness-function selection of chromosomes for the next generation. The dispatching rules are actually contained in the chromosome itself since the GA is not responsible for generating a schedule, but rather for allowing the selection of a dispatching rule.

The fitness function applied to the GA for the advertisement scheduling problem was designed to match the performance measures of the problem. The quality of a solution is measured by the amount of impressions that can be reached on the desired target audiences over all jobs and machines, and the minimal tardiness of the jobs.

The implemented algorithm applied a uniform crossover on two solutions selected with tournament selection. No elitism was applied, and several different mutation rates were tested. The applied mutation randomly recreates the entire schedule for one single machine.

D. Evaluating the heuristics against each other

The heuristics are inherently comparable because the same representation form for the solution was selected for all of them. The quality of all results was calculated using the evaluation function available in the Genetic Algorithm. This had to be done because Distribution Rules do not calculate the Quality of a solution.

TABLE I
APPLIED ALGORIHMS AND PARAMETERS

Algorithm	Parameter Name	Parameter Values
Distribution Rules	-	-
Iterated Local Search	kick rate	$\alpha = \{1, 5, 10, 20\}(\%)$
	iterations	$it = \{100, 1000\}$
	local search steps	$l = \{100, 1000, 10000\}$
Genetic Algorithm	elitism	$e = \{0\}$
	generations	$g = \{1000\}$
	mutation probability	$\alpha = \{1, 5, 10, 20\}(\%)$
	population size	$p = \{100, 200, 1000\}$

The evaluation was done with test data as opposed to using real jobs and machines. This was done due to the fact that only very few machines were available at the time of writing, and the expected growth of the system amounts to at least 1000 machines per year. Thus performance was also an issue, and it could not readily be tested with the machines available.

The generated test data was entirely randomly generated with constraints based on observations made on available real data. For example, the amount of impressions a machine can generate daily is random in between the realistic minima and maxima observed, and includes some outliers as observed in real data.

III. RESULTS

All Algorithms were tested and compared using the same generated testdata. The tests were conducted in problem spaces of different sizes. The size of a problem space is defined by the amount of machines $M = \{1, 10, 100\}$ and Jobs $J = \{1, 10, 100, 500, 1000\}$ in the problem set. Each test set consists of a subset of the next-larger one. The test parameters can be seen in I. Test runs were conducted with all available permutations of the parameters with the exception of 1000 iterations x 1000 local search steps in ILS.

The theoretical optimal quality of the perfect solution is 0, meaning that the entire target audience of the job was met, and no tardiness occurred. The more tardiness occurs or target audiences are missed, the higher the value of the quality becomes. The real optimum of each test-case is unknown since the tests were auto-generated rather than using benchmarks or pre-made tests.

For the distribution rule, only one single run configuration (prioritize projects by budget, assign to machines by amount of impressions generated) was tested (figure 6).

The ILS was tested with different kick rates $\alpha = \{1, 5, 10, 20\}(\%)$, meaning that different percentages of the solution were modified. Furthermore different amounts of iterations $it = \{100, 1000\}$ and local search steps $l = \{100, 1000, 10000\}$ were tested. The results can be seen in figures 1, 2, 3 and 4 each figure showing results for one selected kick rate. Figure 5 shows the final comparisons between the best parameters of each group.

ILS works best with a local search rate of 10000 and 100 iterations. It is not surprising that the execution with the most

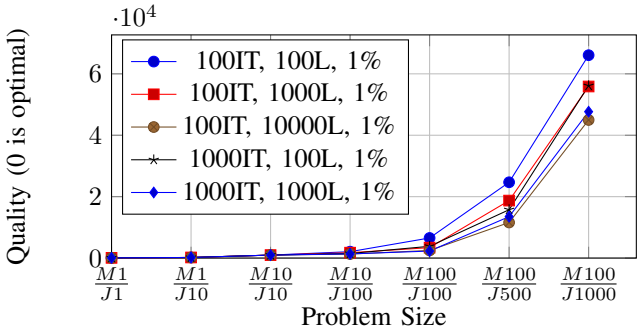

Fig. 1. Quality in differently sized search spaces for Iterative Local Search kick rate $\alpha = 1\%$

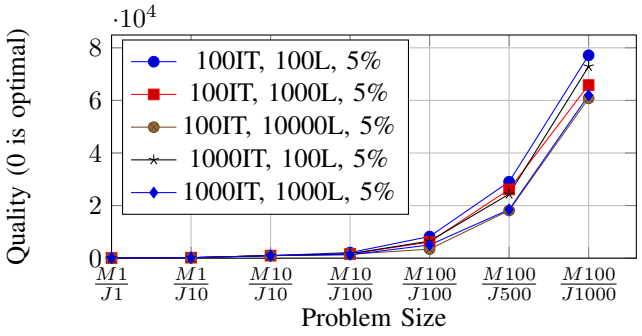

Fig. 2. Quality in differently sized search spaces for Iterative Local Search kick rate $\alpha = 5\%$

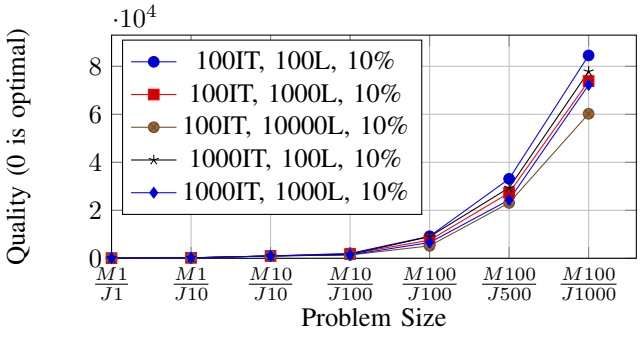

Fig. 3. Quality in differently sized search spaces for Iterative Local Search kick rate $\alpha = 10\%$

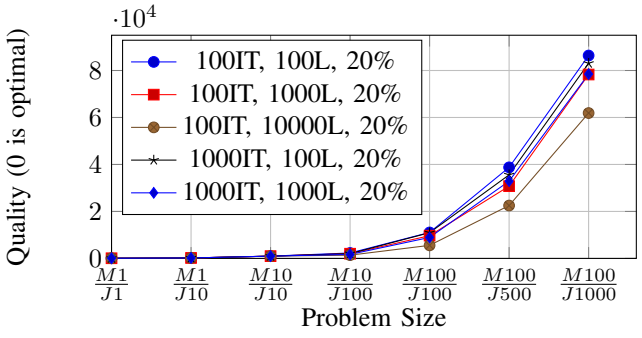

Fig. 4. Quality in differently sized search spaces for Iterative Local Search kick rate $\alpha = 20\%$

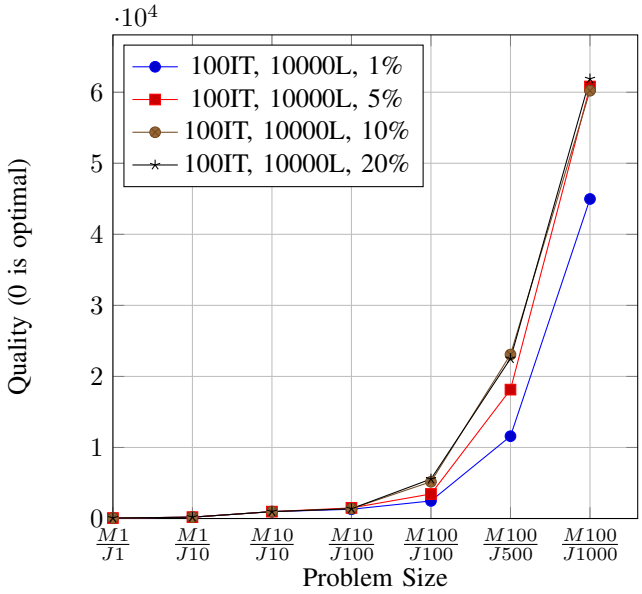

Fig. 5. Quality in differently sized search spaces for Iterative Local Search over different kick rates

operations performs best. However, since the runs with 1000 local searches to 1000 iterations perform only slightly worse, this indicates that more local searches than iterations should be performed.

No matter which combination of runs was taken, a 1% kick rate always vastly outperforms similar runs with higher kick rates (figure 5). This can be attributed to the fact that kicking 1% of the solution moves it far enough out of the local optimum to find a new one, while kicking more percent results in a loss of information from the local search steps and thus performs worse. This result is reinforced by the fact that each lower kickrate outperforms each higher one.

The genetic algorithm runs were performed with population sizes $p = \{100, 200, 1000\}$ and mutation rates $\alpha = \{1, 5, 10, 20\}(\%)$. All runs were conducted over 1000 generations. The results in table II show that any mutation or population size is an insignificant change. However, overall the best-performing runs had a 20% mutation rate, which indicates that machine schedules are independent from each other and thus stochastic search methods would be better suited for application in the problem space.

Figure 6 shows a comparison between the three different applied heuristics. For GA and ILS the solutions with the best parameters were selected respectively.

It was to be expected that the DR as construction heuristics are outperformed by the two optimization heuristics. The fact that ILS vastly outperforms GA reinforces the suggestion that the problem is better suited for stochastic methods, as the high-mutation rates on GA previously suggested.

The vast difference of quality between the two is surprising, since GA is usually only slightly outperformed by LS related methods such as Tabu Search. This discrepancy may stem from the smaller problem sizes usually looked at in benchmarks and

TABLE II
QUALITY OF THE RESULTS IN GA ACCORDING TO MUTATION RATE $\alpha(\%)$, POPULATION SIZE p IN DIFFERENT PROBLEM SIZES SORTED BY MACHINES M AND JOBS J

α	p	$\frac{M1}{J1}$	$\frac{M1}{J10}$	$\frac{M10}{J10}$	$\frac{M10}{J100}$	$\frac{M100}{J100}$	$\frac{M100}{J500}$	$\frac{M100}{J1000}$
1	100	66	204	969	2948	2128	30160	78249
1	200	66	201	969	2686	2262	30508	78816
1	1000	66	194	969	2110	2342	31251	78471
5	100	66	199	969	2584	1664	28898	76758
5	200	66	194	969	2247	1997	29871	78603
5	1000	66	194	969	2180	1806	30621	78185
10	100	66	196	969	2079	1098	28212	76405
10	200	66	194	969	2417	1573	29310	77501
10	1000	66	194	969	1941	2104	28302	78145
20	100	66	196	969	2021	1200	28599	75996
20	200	66	182	969	2206	1412	27913	75550
20	1000	66	194	969	2494	1921	28592	77165

where similar to the findings here. Small problem spaces show only small differences in quality. [8]

IV. DISCUSSION

The results adhere to the no free lunch theorem [20], which states that no one algorithm is superior over all problem-spaces, showing that ILS outperforms this particular instance of the job shop problem.

The results also show similar to other findings that construction heuristics are outperformed by optimization heuristics. The ILS can be added to the local search methods that outperforms GA, which already holds true for Tabu Search and Variable Depth Search. [8]

V. OUTLOOK

Since DR, GA, and the ILS all already exist and have been applied to the problem space, it stands to be reasoned that a hybridization would lead to interesting results. This has been done already with promising results [19], and could be a promising area of research on this particular problem.

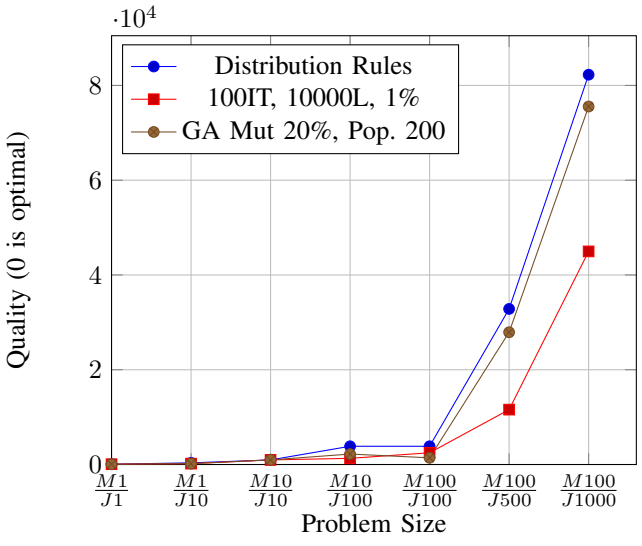

Fig. 6. Quality comparison between DR, GA, and ILS

The discrepancy between the findings here and literature concerning the performance on large problem spaces demands further investigation. It is unclear if ILS really vastly outperforms the GA and Distribution rules in large problem space or if this is only true for this particular problem space. Testing the algorithms with the most basic agreed upon version of job shop scheduling [1] and possibly existing benchmarks upscaled to large problem sizes ($\geq$ 100 machines and $\geq$ 100 jobs) should yield answers in this context.

REFERENCES

[1] Cheng R., Gen M., Tsujimura Y.: A Tutorial Survey Of Job-Shop Scheduling Problems Using Genetic Algorithms - I. Representation Computers ind. Engng. 30, 983-997 (1996)

[2] Cheng R., Gen M., Tsujimura Y.: A Tutorial Survey Of Job-Shop Scheduling Problems Using Genetic Algorithms, part II: hybrid genetic search strategies Computers ind. Engng. 36, 343-364 3 (1999)

[3] Tay J. C., Ho N. B.: Evolving Dispatching Rules Using Genetic Programming For Solving Multi-Objective Flexible Job-Shop Problems Computers ind. Engng. 54, 453-473 (2008)

[4] Dorndorf U., Pesch E.: Evolution Based Learning In A Job Shop Scheduling Environment Computers Ops Res. 22, 25-40 (1995)

[5] Kaban A. K., Othman Z., Rohman, D.S.: Comparison of Dispatching Rules in Job-Shop Scheduling Problem Using Simulation: A Case Study Int. J. Simul. Model. 11, 3, 129-140 (2012)

[6] Holthaus O., Chandrasekharan R.: Efficient dispatching rules for scheduling in a job shop Int. J. Production Economics 48, 87-105 (1997)

[7] Ochoa G., Harvey I., Buxton H.: Optimal Mutation Rates and Selection Pressure in Genetic Algorithms Proc. Gen. Evo. Compu. Conf. - GECCO (2000)

[8] Vaessens R.J.M., Aarts E.H.L., Lenstra J.K.: Job Shop Scheduling by Local Search INFORMS J. Comp. 8 Iss. 3, 302-317 (1996)

[9] Dong X., Huang H., Chen P.: An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion Comp. & Op. Research 36, 1664-1669 (2009)

[10] Dong X., Chen P., Huang H., Nowak M.: A multi-restart iterated local search algorithm for the permutation flow shop problem minimazing total flow time Comp. & Op. Research 40, 627-632 (2013)

[11] Watson J.-P., Beck J. C., Howe A. E., Whitley L. D.: Problem difficulty for tabu search in job-shop scheduling Artificial Intelligence 143, 189-217 (2003)

[12] Balas E., Vazacopoulos A.: Guided Local Search with Shifting Bottleneck for Job Shop Scheduling Mgmt. Sci. 44, 262-275 (1998)

[13] Rajabinasab A., Mansour S.: Dynamic flexible job shop scheduling with alternative process plans: an agent based approach Int. J. Adv. Manuf. Technol. 54, 1091-1107 (2011)

[14] Sabuncuoglu I., Bayız M.: Analysis of reactive scheduling problems in a job shop environment Eur. J. of Operational Research 126, 567-586 (2000)

[15] Park J., Nguyen S., Zhang M., Johnston M.: Evolving Ensembles of Dispatching Rules Using Genetic Programming for Job Shop Scheduling Genetic Programming 9025, 92-104 (2015)

[16] Pezzella F., Morganti G., Ciaschetti G.: A genetic Algorithm for the Flexible Job-shop Scheduling Problem Comp. & Operations Research 35, 3202-3212 (2008)

[17] Park B. J., Choi H. R., Kim H. S.: A hybrid genetic algorithm for the job shop scheduling problems Comp. & Ind. Engng. 45, 597-613 (2003)

[18] Joo C. M., Kim B. S.: Hybrid genetic algorithms with dispatching rules for unrelated parallel machine scheduling with setup time and production availability Com. & Ind. Engng. 85, 102-109 (2015)

[19] Huang J., Ser G. A.: A dispatching rule-based genetic algorithm for multi-objective job shop scheduling using fuzzy satisfaction levels Comp. & Ind. Engng. 86, 29-42 (2015)

[20] Wolpert D.H., Macready W. G.: No Free Lunch Theorems for Optimization IEEE Trans. Evol. Comp. 1 No. 1, 67-82 (1997)