Data Mining Parameters' Selection Procedure Applied to a Multi-Start Local Search Algorithm for the Permutation Flow Shop Scheduling Problem*

Nikolaos Makrymanolakis
Decision Support Systems Laboratory
School of Production Engineering & Management
Technical University of Crete
Email: nikos@nmakry.gr

Magdalene Marinaki
Computational Mechanics & Optimization Laboratory
School of Production Engineering & Management
Technical University of Crete
Email: magda@dssl.tuc.gr

Yannis Marinakis
Decision Support Systems Laboratory
School of Production Engineering & Management
Technical University of Crete
Email: marinakis@ergasya.tuc.gr

***Abstract*— In this paper, a new metaheuristic algorithm is developed, suitable for solving combinatorial optimization problems, such as the job shop scheduling problems, the travelling salesman problem, the vehicle routing problem, etc. This study focuses on permutation flow-shop scheduling problem. The proposed algorithm combines various techniques used in local search. As various elements of the proposed algorithm may be tuned, a systematic data mining procedure is followed and utilizes data from a number of executions in order to build models for the suitable parameterization for every problem size. The results, using the model suggested parameter combinations, are presented using benchmark instances for the permutation flow-shop scheduling problem from the literature. The results show that the followed parameter control procedure improved vastly the efficiency of the proposed algorithm.**

Keywords—combinatorial optimization; flow-shop scheduling; parameter control; data mining; threshold accepting; NEH; Path Relinking;

I. INTRODUCTION

Constructing an intelligent metaheuristic or evolutionary algorithm for combinatorial optimization problems usually isn't sufficient in order to be efficient. Crucial part of the algorithm's development procedure is the finding of the efficient parameters for the algorithm operation, as parameter selection greatly affects algorithm's performance [1, 2, 3]. In this paper, a Data Mining-based parameters' selection procedure is demonstrated on a newly developed algorithm for the Permutation Flow Shop Scheduling problem. The used parameter optimization procedure, vastly improved the proposed algorithm's performance in terms of solution quality compared to the solutions derived with fixed parameters. Although the procedure is presented on a specific algorithm, that demonstration can be used as a guide to most metaheuristic and evolutionary algorithms for combinatorial optimization problems.

Permutation Flow Shop Scheduling Problem (PFSP) is a heavily studied combinatorial optimization problem. The problem consists of n-jobs that have to be processed, by the same order, in each and every of the m-machines. Each job requires a known amount of process time on each machine which varies from job to job and from machine to machine. A machine can process only one job at a time and a job can be processed by only one machine at a time. No interruption on processing of a job is allowed and it is not allowed for a job to enter for processing in the same machine twice. The goal is to find the sequence that the machines will process all the jobs in order to minimize the total process time (the C_{max} or makespan), i.e., the time that is required for the last job to finish on the last machine. A more detailed presentation of PFSP and a quick review on the approaches for the problem can be found in [4].

In this paper, a new metaheuristic optimization algorithm for the PFSP is proposed, based on Threshold Accepting (TA) concept, as originally proposed by Dueck and Scheuer [5]. The main idea behind TA is that in order to avoid trapping in local minima, not only better solutions than the current solution are accepted, but also worse solutions are accepted, if their values are within a predetermined threshold. TA is somehow a deterministic approach of Simulated Annealing (SA) [6] where the acceptance or not of a worse solution is stochastic. The proposed algorithm also incorporates other elements used in optimization algorithms, such as Path Relinking, multiple solutions, etc. Also, the proposed algorithm uses problem specific heuristics, like the NEH algorithm, an approach that is common to many optimization efforts for the PFSP. These elements are presented in detail in the relevant session.

The user has control over three parameters on the proposed algorithm. The first parameter (p_1) is the number of solutions that the algorithm uses which obviously should be at least one. The second parameter (p_2) is the number of iterations of the local search within the neighborhood of each solution. The third parameter (p_3) is the number of periods, the number of process repeat for every solution. The proposed algorithm tested on a common in literature [12] set of problem instances (PI) of various sizes (number of jobs, number of machines - *n x*

*This work has been sponsored by RESEARCH PROJECTS FOR EXCELLENCE IKY/SIEMENS

m). A Data Mining procedure was applied, in order to determine which set of parameters produced better results in terms of optimality of makespan at the shortest time. The proposed algorithm was executed with various settings of the parameters at the different PI sizes, and the result of each execution, characterized in terms of makespan and execution time of the algorithm. Then, by using Decision Trees, the parameter configuration that yields to the most efficient results was selected.

The above parameters' optimization procedure improved vastly algorithm's performance as denoted by results. Data Mining optimized the parameters for each PI size, by combining feedback by all executions in same PI sizes. This optimization procedure can be used in various optimization algorithms to improve their efficiency by selecting the optimal parameters.

II. PFSP Characteristics and Problems Set

As described, solving the PFSP is to find a jobs' sequence that minimize the makespan. As each PI is actually a set of n jobs, then, a solution S is a representation of the sequence *S={1, 2, 3, …, n}* in which the jobs will be executed/processed by the m machines. So, every solution is represented with such a string under the constraints that each job exists exactly once in the string and the string contains all the jobs.

PFSP is a well known strongly NP-hard problem for 3 or more machines, as shown in [7]. There are algorithms that use a constructive approach for a solution, meaning they start the solution string with one job, and step by step they add another and another job in order to have all the jobs in the solution. For 2 machines, there is Johnson's constructive heuristic algorithm [8] that finds the optimal solution in O(n log n). This algorithm may also be used for solving optimally the 3 machines PFSP, but under the restriction that the maximum process time for a job in the second (middle) machine should be less or equal to the minimum process time for a job in first or third machine. If that restriction is not met, Johnson's algorithm cannot be applied for optimally solving a PFSP with more than 2 machines. In 1983, Nawaz, Enscore and Ham propose [9] a constructive heuristic algorithm (NEH) for the general PFSP that leads not to optimum but to a sufficient close to optimum solution for many PI. Although the NEH algorithm does not produce immediately the optimum solution, it is used by many researchers as a means to produce an initial solution for their algorithms, as indicatively in [10, 11]. Having a good initial solution is a good starting point for every optimization algorithm, in order to reach a better or even the optimum solution on a PI. The proposed algorithm uses the NEH algorithm, as described later in the text.

Besides the constructive approach to get a solution for the PFSP, there is the improvement approach. Under the improvement approach, one has a complete and valid solution string, with all n jobs, and creates a new solution by permutations of the jobs in solution string. New solutions derive by interchanging the position (processing order) of 2 jobs, or by removing a job from its position in the sequence and inserting it to another position. Most metaheuristic and evolutionary algorithms for the PFSP use the improvement approach, starting from single or multiple complete solution strings. The proposed algorithm uses the improvement approach, as described later in the text.

In order to test the efficiency of a proposed algorithm for flow shop scheduling problems, Taillard proposed in 1993 [12] a set of benchmark PI. There are different sets with 20, 50 100, 200 and 500 jobs and with 5, 10 and 20 machines. There are 12 sets with 10 instances in each set. The 12 sets are: 20 x 5 (20 jobs x 5 machines), 20 x 10, 20 x 20, 50 x 5, 50 x 10, 50 x 20, 100 x 5, 100 x 10, 100 x 20, 200 x 10, 200 x 20 and 500 x 20. Most algorithms in literature for the PFSP use the Taillard's problem set for testing purposes.

III. The Proposed Algorithm

This session presents the elements of the proposed algorithm.

A. Number of solutions and Path Relinking

Generally, TA is considered a single solution approach, meaning that there is a single starting and working solution. The value of derived new solutions, are compared with the value of that single solution. Other algorithms, like Genetic Algorithms or Particle Swarm Optimization Algorithms, use a finite number of solutions that cooperate as a group in order to track down the optimum.

The proposed algorithm has the option to be a multi-start local search optimization algorithm, as the number of initial solutions is user defined. The starting solutions are not in some kind of cooperation in order to track down the optimum, as happens in Genetic or Particle Swarm Optimization algorithms. The solutions are always individually processed, thus for instance if the algorithm is executed with 10 solutions, it is *almost* the same as to execute the algorithm 10 times using one solution. Actually it is not the same, because using multiple solutions there is a gain by the overhead to initiate the algorithm (loading the data table, create the initial solution, etc.) So, there is a gain in execution using multiple solutions, as opposed using a single solution in each execution. The general idea behind multi-start algorithms is to achieve a better distribution initially in the solutions' space and therefore increasing the chance escaping from local minima [20].

Another possible gain by using multiple solutions comes from Path Relinking (PR) [13]. PR starts from a solution and gradually constructs a path that leads toward to another solution. The general idea behind PR is that possibly a better solution may exists in the path that connects two existing solutions. The path between the two solutions is generated by altering the attributes of initial solution in order to gradually match to the target solution. The proposed algorithm incorporates PR between the current best solution and the current group of solutions that are in progress. If only one solution is used for the process, then, that solution is Path Relinked to the current best solution.

At any case, the current best solution is hold separately, and it doesn't count to the number of solutions. On the proposed algorithm, the number of solutions is user defined, as parameter p_1.

The following pseudo code represents the Path Relinking part of the proposed algorithm:

Algorithm Path Relinking
Repeat
    If a different job is found in initial solution then
        Locate that job in target solution
        Exchange positions of jobs in initial solution
        If a better solution is found, keep it
    End if
Until the end of initial solution string

B. Neighborhood and acceptance criteria

The essence of a neighborhood is the solutions that may be derived by a given solution string. The proposed algorithm uses a stochastic neighborhood approach. Given a solution, a random job in the solution is selected, is removed from its position in the sequence and is inserted at another position in the solution string. Insertion preferred over exchange of jobs' positions, as some researchers recommend that insertion is more effective than exchange [14].

For each new solution, its makespan is calculated. If it is better than the current best solution s^*, then, the new solution is considered as the best solution and the neighborhood of the new solution is searched. If the new solution is better than the solution that initiated the neighborhood or even if it is not worse than the starting solution plus the threshold, then the neighborhood of the new solution is searched. With this mechanism, different places of solutions space are examined and also entrapment to local minima is avoided, as it would possibly happen if a constant descent strategy was followed.

As already mentioned, the whole process is repeated for a number of periods, controlled by user defined parameter p_3. The value of the threshold is not constant in every period. At the beginning, the threshold equals to p_3 (if user requests 5 periods then the threshold is 5), which means that a new solution is accepted if it not worse than current solution's makespan plus p_3. At next repetition the remaining periods decreased by one, and so does the threshold. This implementation is based on the strategy, derived by Simulated Annealing [15]. Under this strategy, at the beginning the algorithm is more receptive of worse solutions, in order to move faster to different places of solutions space, but as the algorithm converges, it becomes more and more rigid.

A simplified pseudo code representation of the above procedure is the following.

Algorithm Neighoborhood search
For every solusion s
    For p_2 repetitions
        Create a new solution s' from s
        If s'< s then s*=s' and s=s'*
        If s'<(s+p_3) then s=s'
    End For
End For

C. Initial solutions

The first initial solution for the proposed algorithm comes from the NEH algorithm [9]. If the user defined number of solutions (p_1) is more than one, then, the rest solutions are derived from NEH solution by permutations in the percentage of 25% of jobs number. Percentage was preferred offer a fixed number, because for instance 10 permutations are sufficient for a 20 jobs PI string, but totally insufficient in terms of placement to different places of solutions space, for a 500 jobs PI. As in literature NEH derived solution is considered a good starting point, it wasn't desirable to move far away from it, as that would be equal as having completely random initial solutions.

D. Total algorithm operation

So far, the basic elements of proposed algorithm for the PFSP were described. The following pseudo code presents the total operation of the proposed algorithm.

Algorithm for the PFSP
Create initial p_1 solutions
s= the best of p_1 solutions*
For p_3 repetitions
    Do Neighborhood search
    Do Path Relinking
End For
Return s as best solution*

The proposed algorithm implemented in standard C-99, compiled by GNU C compiler v. 3.4.3, under 32bit version of Windows 7 on a Intel core 2 duo, 2 GHz CPU. The same computer is used for all executions and testing (single core executions).

IV. Parameter Control Issues

As mentioned, crucial part of the algorithm's development procedure is the optimal parameter control as parameters greatly affect algorithm's performance [1, 2, 3]. Also, as shown in studies like [16], the characteristics of a problem also have great influence on algorithm's performance. So, the general situation is that there is not a single parameter set for an algorithm in order to be performed equally efficiently for every problem.

Efficiency in this paper has two requirements. The first is to find the best solution for every problem, the optimum if possible. The second requirement is to find that solution, in the shortest possible time. The benchmarks for the PFSP are the Taillard's set of scheduling PI that vary in size in terms of the number of jobs and of the number of machines. Thoroughly testing using the proposed algorithm over the Taillard's set of PI indicated that different parameters had significant effect on the efficiency of the algorithm.

Table I illustrates a specific numerical example, on Taillard's PI #71, with 100 jobs and 10 machines, using four different parameter combinations. The fourth parameter combination gave the best solution, but it is the most costly in time among the other executions. The third parameter combination gave a close to best solution, in significantly less time. The second execution gave a solution close to the previous one, but it took almost the double time. And comparing the first and third execution, the first is dominated by third in both of terms of makespan and time. Executions with different parameter combinations on Taillard's set of PI gave varying results, in terms of efficiency and inefficiency of parameter combinations.

Considering all the above, two crucial questions arise. First: By which parameter combination the best makespan is found in the shortest time? Second: Is there a parameter combination with the same efficiency in all problem sets?

After a few hundred of executions, with various parameter combinations on various PI, is realized that different PI sizes require different parameter combinations to achieve efficient solutions. Thus, there is a strong indication that the answer to the second question is most probably negative.

TABLE I. INDICATIVE EXECUTIONS ON TAILLARD PROBLEM #71

No	p_1	p_2	p_3	Makespan	Time (seconds)
1	8	500	5	5822	1.213
2	5	1000	5	5801	1.300
3	1	4000	5	5800	0.713
4	1	72000	5	5785	9.070

More solutions (p_1), more repetitions (p_2) and more periods (p_3), certainly increase execution time. The increase of p_1 may lead to a better makespan, or at least there is absolutely no one reason that may lead to a worse makespan. But increasing p_2 and p_3 infinitely may deteriorate the makespan in addition to the deterioration of execution time, producing totally inefficient solutions. Also, if a parameter combination may produce the same makespan in less time this combination dominates over other parameter combinations that produce similar makespan in more time. All these considerations are for each PI size, so each problem size has to be processed individually.

In order to realize the significance of parameter setting, on Taillard's PI #112 (500 x 20), there were executions with makespan approximately 26870, using various parameter combinations, in 73, 101, 147, 205, 288, 515, 588, 816 and 8525 seconds. So, why to wait 2.5 hours to get an equally good makespan that you can get in 1-2 minutes, only by choosing a more appropriate parameter set? To estimate efficient parameter combinations, which lead to good makespan without wasted execution time, a Data Mining based procedure is used.

V. DATA MINING UTILIZATION

Data Mining (DM) is a process used to seek interesting or useful patterns and relationships in huge amount of data. It is a procedure to create models from available data, and apply these models to analyze similar situations [17]. Data Mining methods and techniques, explore and analyze in automated –or usually in semi-automated- way data sets, in order to track down unknown or hidden relationships or trends that govern these data sets. DM is a process of description and prediction. Description is about processing of existing data, and prediction is about applying the attributes derived from processing to new data, new situations, with yet unknown outcome [18]. In other words, is an effort from available data to estimate the outcome of other situations, for which there are no available data.

DM projects usually follow industry standard CRISP-DM (Cross-Industry Standard Process for Data Mining), developed by industry at the end of '90es. CRISP-DM incorporates six steps for the DM process, namely [19]:

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. Deployment

A complete DM project was deployed in order to estimate the proper parameter set for every PI size.

It should be noted that the described DM procedure took place outside of the proposed algorithm, and it is not part of the algorithm itself. It is not carried out adaptively, as part of the algorithm, but it uses data from algorithm execution, that are processed separately. The potential of embedding a similar DM procedure inside an optimization algorithm is part of intended research for the future.

A. Business and Data Understanding

The proposed algorithm processed the Taillard's problem set using different parameter combinations. For each execution was recorded:

- The number of jobs (from problem size)
- The number of machines (from problem size)
- The number of solutions (p_1)
- The number of repetitions (p_2)
- The number of sets (p_3)
- The produced makespan
- The execution time

In order to reduce the number of executions, all testing done using 5 periods (p_3=5), as a significant number of tests indicated that increasing p_3 to more than 5, didn't significantly improve the makespan and it certainly increased execution time. So, actually testing involved 2 user-controlled parameters, p_1 and p_2. Also, since it was not practical to run the tests using every integer combination of *p1* and *p2*, increments according to Fibonacci sequence were picked. Thus, p_1 took values *{1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144}* and p_2 the values *{500, 1000, 1500, 2500, 4000, 6500, 10500, 17000, 27500, 44500, 72000, 116500}*. At least 3 PI from each of 12 Taillard's PI sizes (even the biggest ones) were randomly selected for testing executions, for every combination of the above values for p_1 and p_2 (132 executions). In smaller PI, all problem sets were executed for testing. Each combination executed 10 times, in order to record the best, the mean and the variance of makespan, as the proposed algorithm contains stochastic elements.

B. Data Preparation

Having a lot of thousand of records from executions, the next step is the preparation of the data. Every record was characterized in terms of makespan quality and execution time.

Starting by makespan, for every problem set there is a Best Known Solution (BKS) [4]. Three more or less equal sized groups created, according to the difference of record's makespan to BKS. The more close records are characterized as Very Good (VG), the next group as Good (G) and the most distant group as Bad (B). For the execution time, three groups also created. But in case of execution time, there are not benchmark times, so comparable times used according to fastest and slowest execution for each problem set. As Very Fast (VF) were characterized the records with execution time less than *Fastest Time + (Slowest-Fastest)/100*. Records with execution time more than *Fastest Time + (Slowest-Fastest)/10* were characterized as Slow (S) and the remaining records were characterized as Fast (F).

For some problem sets it was necessary to adjust the above limits in order to have a meaningful population for each category. It was especially significant to have a population of records characterized as VGVF, because this is the goal to estimate the DM procedure, so that attribute had to be demonstrated at the training phase. Finally, some clearance on the data was made, by removing inconsistencies due to stochastic nature of proposed algorithm. For instance, if a record with *p1=2* and *p2=500* gave a VGVF solution and another record with *p1=8* and *p2=500* gave a BADVF solution, the second record was removed, as it is not consistent to use more solutions and get a worse makespan. Of course, if the second record was VGF it was accepted, as more solutions should lead to bigger execution time. This data clearance improved data quality for training the model.

C. Modelling

In order to execute the DM process, RapidMiner Studio version 6.4 software was used and a Decision Tree model was created. Previously prepared data were used for training of the RapidMiner model. RapidMiner software possesses an internal parameter optimization feature. By using that feature, which is a user controlled procedure, the user can estimate the most suitable parameters for the RapidMiner model. These parameters are data set specific, and on a different data set a new parameter optimization for the RapidMiner model should take place. Of course, there is always the option for the RapidMiner user to accept the default parameters for every model, but parameter optimization will produce better results. After the parameter optimization procedure, the following parameters for the Decision Tree process were used:

- Criterion: Information Gain
- Maximal Depth: 11
- Apply Pruning: ON
- Confidence: 0.25
- Apply Prepruning: ON
- Minimal Gain: 0.0
- Minimal Leaf Size: 5
- Minimal Size for Split: 11
- Number of prepruning alternatives: 100

D. Evaluation

The next step is the evaluation of the model. The evaluation applies the model to training data and compares the actual quality attribute with the quality attribute the model estimates for every record. The total precision of Decision Tree model is estimated at 64.89% (±1.18%), according to RapidMiner software. In Table II there are the percentages of recall and precision for every class.

TABLE II. PREDICTION AND RECALL RATES OF DECISION TREE MODEL

Predicted as	%	Recall as	%
pred. BADVF	61.95%	true BADVF	70.29%
pred. VGVF	47.84%	true VGVF	51.98%
pred. VGF	68.02%	true VGF	82.01%
pred. BADF	55.22%	true BADF	22.22%
pred. VGSLOW	88.99%	true VGSLOW	84.73%
pred. BADSLOW	0.00%	true BADSLOW	0.00%
pred. GVF	54.16%	true GVF	41.83%
pred. GF	58.33%	true GF	32.34%
pred. GSLOW	33.33%	true GSLOW	4.35%

E. Deployment

After creating the model, it is applied for attribute estimation on each combination of p_1 and p_2. The application of the model is very fast, so there is actually no barrier in order to be as dense as preferred in parameter increment. An estimation for 14 increments of p_1 (1 to 120) and 37 increments of p_2 (100 to 100.000) was requested. The model produced confidence estimation for the 9 possible attributes (VGVF, VGF, VGSLOW, GVF, GF, GSLOW, BADVF, BADF, and BADSLOW).

Then it follows the interpretation of the predicted attributes, which is not a straightforward procedure, as most of the times the model didn't designate one single combination with a distant best confident value. So, a human decision is required to select one or more parameter combination for every problem set size. Obviously, the most preferred combinations are the ones that the model predicts as VGVF (Very Good/Very Fast), meaning that they will probably give a minimal makespan, with low execution time. In some cases, VGF (Very Good/Fast) combinations also considered, especially in problem set groups that the model didn't propose any VGVF combinations. Table III presents the most efficient parameter combination that finally selected for every problem set size.

It is obvious in Table III that to smaller PI sizes more solutions and fewer repetitions are preferred, while in bigger PI

sizes, the preferable number of solutions decreases and the number of repetitions increases.

The data in Table III are pretty consistent with the intuition about the trend for each parameter. Increasing the number of solutions (p_1) lowers the makespan, but increases the execution time. As PI size increases, a big number of solutions certainly may increase the probability for lower makespan, but that makespan gain does not balance with the increase in execution time. So, parameter combinations with many solutions are unlikely to produce VGVF solutions in bigger PI. Additionally, as number of jobs increases in problem sets, more repetitions (p_2) are required in order to achieve sufficient neighborhood exploration. So, to bigger PI sizes, more repetitions are more likely to contribute to an efficient search and by extension to an efficient solution. On the contrary, smaller PI sizes are explored more efficient with a population of solutions, without the need of many repetitions as their neighborhoods are smaller.

TABLE III. MOST EFFICIENT PARAMETER COMBINATION FOR EACH PROBLEM SET SIZE

Size	Solutions (p1)	Repetitions (p2)
20x5	75	850
20x10	83	1000
20x20	35	3000
50x5	45	3000
50x10	4	30000
50x20	6	30000
100x5	2	55000
100x10	2	55000
100x20	2	55000
200x10	1	90000
200x20	1	90000
500x20	2	90000

VI. PROPOSED ALGORITHM EXECUTION DATA

The proposed algorithm executed all Taillard's problem set 10 times each, using the parameters from Table III, and $p_3=5$ for all executions. The repeated execution was necessary, as the proposed algorithm contains stochastic elements, so it is not enough to get a good solution in only one execution, but it is important for the robustness of the algorithm to produce good solutions in most of the executions. From the 10 executions of every problem set, each individual makespan is denoted as C_j, the minimum makespan is denoted as C_{min}, the maximum makespan is denoted as C_{max} and the average makespan of all 10 executions is denoted as $C_{aver}=\frac{\sum_{j=1}^{10} C_j}{10}$. The results of every problem set is compared with BKS of the set, and grouped per PI size. The deviation of minimum, maximum and average makespan per size group is calculated by (1), (2) and (3) respectively.

$$\omega_{min} = \frac{\sum_{i=1}^{10}\left\{\frac{Cmin_i - BKS_i}{BKS_i}\right\}}{10}\% \quad (1)$$

$$\omega_{max} = \frac{\sum_{i=1}^{10}\left\{\frac{Cmax_i - BKS_i}{BKS_i}\right\}}{10}\% \quad (2)$$

$$\omega_{aver} = \frac{\sum_{i=1}^{10}\left\{\frac{Caver_i - BKS_i}{BKS_i}\right\}}{10}\% \quad (3)$$

Table IV presents the ω_{min}, ω_{max} and ω_{aver} deviations for each group of problem sets and the average time for each group. Bigger deviation is noted in groups 50x20, 100x20 and 200x20, all groups with 20 machines. The specific 30 problem sets are among the hardest in Taillard's problem set, as also noted by other researchers [14], so it is not an unexpected incidence. Generally, as Table IV shows, the proposed algorithm gave impressive results in terms of best, maximum and average makespan for every group of the problem set.

TABLE IV. MEAN DEVIATION OF MINIMUN, MAXIMUM AND AVERAGE RESULT FOR EACH GROUP AND AVERAGE TIME

SET	ω MIN	ω MAX	ω AVER	TIME AVER (seconds)
20x5	0.000	0.624	0.160	1.005
20x10	0.093	0.859	0.478	2.727
20x20	0.142	0.616	0.373	6.916
50x5	0.021	0.266	0.116	4.515
50x10	0.899	1.625	1.277	7.751
50x20	1.527	2.406	1.944	25.294
100x5	0.062	0.187	0.114	6.625
100x10	0.346	0.890	0.592	13.336
100x20	1.826	2.511	2.154	28.515
200x10	0.402	0.667	0.516	21.470
200x20	1.657	2.506	2.069	46.460
500x20	0.890	1.204	1.054	228.821
Total	0.657	1.197	0.905	32.786

The executions using parameters from Table III were not the fastest executions the proposed algorithm produced. There were executions with significant less execution time and some of them produced equally good makespan. For instance, in the first problem set group (20x5) there were executions (with different parameter combination) with 0.07 seconds execution time that tie the BKS. But the selection of parameter combination, using the DM model, was based on the execution of all problem sets, thus, it is affected by the total performance not only by extreme occurrences. Also, as the proposed algorithm incorporates stochastic elements, it is significant getting efficient solutions, in great percentage of executions.

Regarding makespan, in 30 out of 120 PI (25%) the BKS was found. Deviation over 2% occurred in only 6 out of 120 problem sets, and deviation over 1% occurred in 38 out of 120 problem sets. Thus, in 82 out of 120 problem sets the deviation is less than 1%. The proposed algorithm proved fairly efficient even at cases that produced its worst makespan. In 6 PI out of 120 the BKS was found at all executions. The mean deviation of maximum makespan (worst) is 1.197% and the maximum deviation is 3.241% (in a single problem set). Maximum deviation over 3% occurred in 2 problem sets and over 2% in 31 problem sets out of 120. Thus, in 89 out of 120 problem sets, the maximum deviation compared to BKS was less than 2%.

In conclusion, the proposed algorithm, using parameter combination that derived by the described DM process, gave state of the art results, with notably low execution times.

VII. Assessment of DM Procedure

In session IV two questions were raised. First: By which parameter combination the best makespan is found in the shortest time? Second: Is there a parameter combination with the same efficiency in all problem sets?

Regarding the second question, initial indication that better results will come with specific parameter combination for every PI size, proved as true. The parameter combination that DM model proposed in Table III shows clearly that p_1 and p_2 parameters differ significantly from PI size to PI size. So, there is no indication that there possibly exists a single parameter combination that will beat the efficiency of individual parameter combinations for every PI size.

Regarding the first question, Data Mining model proposed efficient parameter combinations, and after processing of those recommendations, the parameter combinations of Table III were produced. The execution results over the Taillard's problem set using those parameter combinations show that efficient solutions achieved, both in terms of makespan value and execution time. Also, efficient results were pretty consistent, as even the worst performance in terms of makespan was sufficient and not significantly distant by BKS.

The DM procedure saved execution time. Increasing the number of solutions could be beneficiary, but the cost of time may be more than the benefit in makespan value (if any). Increasing the number of repetitions -also costly in time- may lead to worse attitude of the proposed algorithm. More repetitions cause broader alteration to initial solution that leads to an expansion of neighborhood search. This may be useful for big PI (many jobs), but not efficient to smaller ones.

The issue of distance from a starting solution, during neighborhood exploration, is important, regarding execution times and PI sizes. A 20x20 PI has 20! solutions, while a 500x20 PI has 500! solutions. Both PI have the same complexity of 20 machines. If 7 seconds are spend for the first and 229 seconds for the last, 33 times more time is dedicated for the bigger PI. But the size of the second PI is $500!/20!=5x10^{1115}$ bigger than the first. Thus, by analogy, to explore the same solution space for the 500x20 PI, as done in 7 seconds for the 20x20 PI, it should be explored for $3{,}5x10^{1116}$ seconds! So, spending 250, 500 or even 10.000 seconds on a 500x20 PI, couldn't lead to the same percentage of exploration as just few seconds do for a smaller PI. By reducing the amount of repetitions and indirectly the amount of execution time, the DM procedure helps to adjust the parameters to meaningful levels, as a bigger increase will not produce better results, to balance the extra cost of execution time.

So the DM procedure directs to the parameter combination that produces an efficient outcome, without unnecessary execution time spending. This is crucial if multiple and fast execution of the solver is required –like in a just-in-time environment- where a good makespan at the lowest possible execution time is a blessing.

VIII. Concluding Remarks

In this session some useful conclusions are recorded about this work and its future prospect. Generally, the contribution of this paper is a new simple, fast and efficient algorithm for the PFSP and other problems, along with a systematic procedure to optimize the operational parameters of the proposed algorithm, which can be generalized and applied to other algorithms also.

A. The proposed algorithm

Implementing an optimization algorithm, require a combination of existing strategies and novelty. The proposed algorithm incorporates known strategies and ideas and combines them in its own manner. Generally it is a simple algorithm, easy to understand and to implement and pretty efficient in terms of optimization and execution time.

The exact same algorithm may be used in any problem that may be solved by rearrangement of points, only by changing the objective function. Such problems are the Travelling Salesman Problem (TSP), the Vehicle Routing Problem (VRP), other scheduling problems like Job-Shop, etc. Of course, in problems other than PFSP, the NEH algorithm cannot be used, but there are corresponding heuristics for many other problems that may be used accordingly for producing the initial solutions. With the exception of NEH algorithm's usage, the proposed algorithm does not take advantage of problem specific characteristics, thus may be used for general combinatorial optimization problems.

Another interesting characteristic of the proposed algorithm is that the user may control some critical parameters of its operation. Besides p_1, p_2 and p_3 parameters, there is the option to user control on additional elements of the proposed algorithm. So, it is possible to replace the Threshold Accepting part with a Simulated Annealing or a tabu search part, or to switch off Path Relinking, just to mention a few modifications. These modifications may also be part of the Data Mining process to determine a mixture of elements in the algorithm that perform better, along with the parameters' estimation.

B. The data mining procedure

The Data Mining procedure that followed for parameter control is systematic, based on solid techniques and principles and it is proposed against arbitrary or deficient procedures that

may be followed for parameter control. The DM procedure allows proceeding in a series of ascertainments.

First, parameter control of an optimization algorithm is a complete Data Mining project, which contains all steps of CRISP-DM model. Second, a great amount of systematic and not random executions is required, in order to acquire knowledge on how the algorithm behaves. Third, it is not realistic to expect that the DM procedure will guide to a single parameter combination, but rather to groups of parameters combinations with specific characteristics, like combinations that are fast, other that are slow but reach the optimum solution, other which are neither optimum nor fast, etc. That clustering of parameter combination offers a lot of alternatives to compare and select. Fourth, the same DM procedure may be used in order to get parameter combinations for different goals, like e.g. combinations that have execution time less than a specific amount of time. By changing the problem in step 1 of CRISP-DM, a different view arises, using the same dataset. Fifth, the dataset is bound with the problem in study. If the problem changes, even slightly, then a new dataset is needed. Sixth, DM models may be used in order to get estimations even for sizes that no primary data are available. For instance, in Taillard's problem set, there are not PI sized 500x5. DM model proposes that using 2 solutions and 40.000 repetitions is a parameter combination that would produce VGVF solutions. So, DM models may used in order to make estimations for situations so far unknown, with no real data. Seventh, the DM procedure isn't about picking a single combination from the dataset. It is about combining the dataset, creating unique knowledge from it, and producing something that may do not exist in the dataset. For instance, the combination 6/30000 that used as efficient for 50x20 PI size weren't used in model training dataset. Parameter combination 6/30000 was never executed while creating training data, but that combination emerged through the blending of training dataset, even not existed in the original dataset.

The whole procedure might look tedious and computational expensive, but good parameter sets can be derived with significantly less executions. Also, if in real life applications we are not interesting for some PI dimensions, we can omit these PI from the DM procedure.

C. *Future research*

Applying Data Mining for parameter control on optimization algorithms is mostly unexplored. Most researchers do not mention any specific method that they follow in order to pick the parameters for their algorithms.

The applied DM procedure provided better results by a newly developed algorithm. Following a similar procedure, parameter setting for new and existing algorithms may be optimized, improving their efficiency. This may lead to even better results, using existing algorithmic ideas.

The DM model used is based on Decision Trees. There are many other procedures that could be used for DM modeling, like neural networks. So, a future research should try other modeling for the DM procedure.

Finally, another step could be to use that DM procedure adaptively inside an algorithm, in order to automate parameter control.

REFERENCES

[1] Z. Michalewicz, and M. Schmidt, "Parameter control in practice," in F. G. Lobo, C. F. Lima, and Z. Michalewicz, Eds, Parameter setting in evolutionary algorithms, vol. 54 of Studies in Computational Intelligence, pages 277–294. Springer, 2007

[2] A. E. Eiben, and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," in Swarm and evolutionary computation, 1(1):19–31, 2011.

[3] A. Aleti, "An adaptive approach to controlling parameters of evolutionary algorithms," PhD thesis, Swinburne University of Technology, 2012.

[4] Y. Marinakis, and M. Marinaki, "Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem," Soft Computing, vol. 17, no. 7, pp. 1159–1173, 2013.

[5] G. Dueck, and T. Scheuer, "Threshold accepting: A general purpose optimization algorithm superior to simulated annealing", Journal of Computational Physics, vol. 90, pp. 161-175, Academic Press, 1990.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," Science 220, pp. 671-680, 1983.

[7] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flow-shop and job-shop scheduling," Mathematics of Operations Research 1, pp. 117-129, 1976.

[8] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included", Naval Research Logistics Quarterly 1, pp. 61-68, 1954.

[9] M. Nawaz, E. E. jr.Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," OMEGA The international journal of management science vol. 11, no. 1, pp. 91-95, 1983.

[10] E. Nowicki, and C. Smutnicki, "A fast tabu search algorithm for the permutation flow-shop problem," European Journal of Operational Research 91, pp. 160-175, 1996.

[11] M. Henneberg, and J. S. Neufeld, "A constructive algorithm and a simulated annealing approach for solving flowshop problems with missing operations," International Journal of Production Research, vol. 54, no. 12, pp. 3534-3550, Taylor & Francis, 2016.

[12] E. Taillard, "Benchmarks for basic scheduling problems," European Journal of Operational Research, vol. 64, pp. 278-285, 1993.

[13] F. Glover, and M. Laguna, "Tabu Search," Modern Heuristic Techniques for Combinatorial Problems, C. Reeves (ed.), Blackwell Scientific Publishing, Oxford, pp. 70-150, 1993.

[14] E. Nowicki, and C. Smutnicki, "Some aspects of scatter search in the flow-shop problem," European Journal of Operational Research 169 pp. 654-666, 2006.

[15] E. H. L. Aarts, J. H. M. Korst, and P., J., M. van Laarhoven "Simulated Annealing" in "Local Search in Combinatorial Optimization" E. Aarts, J. K. Lenstra, Eds, John Wiley & Sons, pp. 91-120, 1997.

[16] J-P. Watson, L. Barbulescu, A. E. Howe, and L. D. Whitley, "Algorithm performance and problem structure for flow-shop scheduling," American Association for Artificial Intelligence, 1999.

[17] I. H. Witten, F. Eibe, M. A. Hall, "Data Mining: Practical machine learning tools and techniques," (3rd ed.), Elsevier, 2011.

[18] S. Tuffery, "Data Mining and statistics in decision making", R. Riesco, translator, Wiley, 2011.

[19] M. North, "Data Mining for the masses," Global Text Project (CCA 3.0 License), 2012.

S. W. Lin, K. C. Ying, C. C. Lu, and J. N. D. Gupta, "Applying multi-start simulated annealing to schedule a flow line manufacturing cell with sequence dependent family setup times," International Journal of Production Economics, 130(2), pp. 246-254, 2011.