Proceedings of the 2015 INFORMS Workshop on Data Mining and Analytics
M. G. Baydogan, S. Huang, A. Oztekin, eds.

UNDERSTANDING THE JOB-SHOP SCHEDULING PROBLEM STRUCTURE USING SUPERVISED MACHINE LEARNING

Sadegh Mirshekarian

Department of Industrial & Systems Engineering
Ohio University
Athens, Ohio
sm774113@ohio.edu

Dušan N. Šormaz

Department of Industrial & Systems Engineering
Ohio University
Athens, Ohio

Abstract

Knowledge discovery using machine learning techniques can be an effective way of understanding the influence of Job-Shop Scheduling Problem (JSSP) structure on high-level attributes like makespan. This paper investigates the applicability of supervised machine learning on the issue, and applies techniques like Support Vector Machines and k-Nearest-Neighbors to design discriminative models that classify a JSSP instance as *efficient* or *inefficient*. An efficient instance is one that has an efficient optimal solution, characterized by a lower normalized machine idle time compared to class average. About 90 temporal and non-temporal features were identified and tested, and 10,000 problem instances of different arrangements were randomly generated and solved to serve as training and test data. The results show promising but limited prediction power for non-temporal features and relatively more robust prediction power for temporal ones. The performance was relatively consistent over different problem sizes, although a noticeable degrading trend could still be observed with increasing problem size. A discussion of the lessons learned and future research directions is given in the end.

Keywords: Job-Shop Scheduling, Makespan Prediction, Machine Learning, Support Vector Machines, Knowledge Discovery

Introduction

The Job-Shop Scheduling Problem (JSSP) is well-known to be NP-hard [1] and is classically categorized among the most intractable combinatorial optimization problems considered [2]. There are several variations of JSSP, most of them proven to be NP-Complete [3, 4, 5]. Different solution techniques including exact methods, heuristic and metaheuristics have been suggested for each variation [6, 7, 8]. The variation on which we focus in this paper is the conventional JSSP with no sequence-dependent setup times, no due dates, no preemption, one operation per machine at a time and one machine per job at a time. However, most of the ideas and practices that we develop for this research can be used on other variations with minimal modification.

According to [9], analytical solution techniques for JSSP are plagued by the notorious complexity of the problem, while the real-life dynamic environment in which scheduling is performed adds to the problem and renders most static models irrelevant or impractical. Therefore Operation Research practitioners resort to *dispatching rules*, or heuristics, to solve practical-sized instances in reasonable time (see [10] for a survey). Aytug et al. [9] note that even though some dispatching rules give reasonable results for some problem instances, it is difficult to predict when or for what type of instances they give good results. According to [11], human interaction with these heuristics can often improve their performance. This might be an indication that having good insight into problem structure can go beyond simple heuristics; a major premise of this research. We investigate the use of inductive machine learning techniques in providing more insight into the JSSP structure. See [9] for a classic review of the use of machine learning in scheduling.

Most research on scheduling is about techniques to solve a given instance and find a schedule that is optimal or near-optimal in terms of a certain output (such as makespan, which is defined as the time it takes for all operations to be completed). However, the problem of how to design a JSSP instance to maximize the use of resources is rarely considered. This is an important problem to look at, because JSSP instances can vary greatly in terms of their optimal use of resources. Consider for example the solutions of two instances shown in Figure 1. They have the exact same makespan and number of machines, but assuming that there are no setup times involved, the one on the right has a better use of resources, because more operations are being processed on the machines. The question that we would like to answer is this: of the two given JSSP instances, the optimal solution of which one has a better use of resources? In other words, the optimal solution of which one has the least machine idle time?

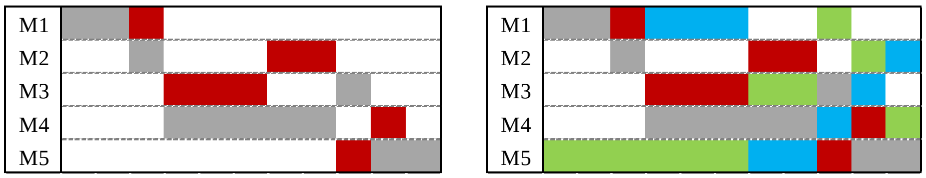

Figure 1: Solutions for two different JSSP instances with five machines

The trivial way of answering these questions is to solve each instance to optimality and then compare the total machine idle time of each of the solutions. Needless to say, this is not tractable for most large instances (number of jobs and machines more than 50), and it may not be practical even for medium-sized ones. It would be interesting if we could find a way to figure out the degree of resource utilization efficiency of a given JSSP instance without having to find its optimal solution. We can use quick and simple heuristics to get an idea of the optimal solution, but we cannot have the actual optimum, or ensure we are closer to it than a desired amount. There are multiple applications for a 'predictor' of optimal schedule efficiency. Here are some of them:

1- We can use the efficiency measure to estimate the optimal makespan of an instance (as explained in the Methodology section), which can in turn be used to guide specially designed heuristics and improve their performance, or to gain insight into an instance and choose the right method to solve the instance from the beginning.

2- We can gen insight into problem structure by analyzing the effect of different features of a given instance on its optimal makespan, and use this insight for making the problem more efficient (if such option exists as changing the problem features). For example it is clear that if one job has a much higher total processing time than other jobs, the optimal schedule will not use the resources efficiently (because one job dominates makespan). The results of this research will hopefully help discover less obvious impact pathways between problem features and makespan.

3- Equivalently, we can verify a relevant claim or conjecture based on expert intuition or insight. For example Taillard [12] conjectured that as the ratio of number of jobs to the number of machines increases, the optimal makespan approaches the following lower bound, in which p_{ji} is the processing time of the *i*th operation of job j, and m_{ji} is the machine responsible for that operation:

$$C_{LB} = \max\{\max_j \sum_i p_{ji}\,,\ \max_i \sum_{j,k|M_{jk}=i} p_{jk}\} \tag{1}$$

Using the results of our research, we hope to be able to verify such correlations by looking at problem instance features.

4- Benchmark JSSP instances are often generated randomly, with trial and error usually being the only way if a certain characteristic is desired for the instances (see [12] or [13]). Using the results of our research, we would be able to use our knowledge of influential features to design benchmark cases which have a special distribution of optimal makespan and/or efficiency.

5- The main concept can be used to predict target parameters of an instance other than makespan.

As this is an ongoing research, we present our roadmap as well as some preliminary results in this paper, and elaborate on them and report the final results in the future. All the problem instances generated and used, as well as all of the MATLAB and Python code written for the various stages of our research are available online or upon request.[1]

Methodology

The first thing we need is a measure of scheduling efficiency that is consistent with our definition of efficiency, and is independent of problem size. Because a more efficient schedule in our definition is one that uses machines better, the measure must somehow be related to total machine idle time. We propose the following metric, which is one plus the total machine idle time per unit processing time of all the jobs.

$$C' = 1 + \frac{\sum_i l_i}{\sum_{j,i} p_{ji}} \tag{2}$$

In this equation $\sum_{j,i} p_{ji}$ is the total processing time of all jobs and l_i is the total idle time of machine *i*. Observe that C' is 1 for a schedule that has no idle time and therefore uses all the machines perfectly, while as schedule efficiency decreases, C' increases. Now note that for any schedule:

$$\sum_{j,i} p_{ji} + \sum_i l_i = C.m \tag{3}$$

Where *C* is the makespan and *m* is the number of machines. Equation 3 is easy to verify. Consider the examples given in Figure 1. $\sum_i l_i$ is the sum of all the white spaces because they denote machine idle times, and when this value is added to the sum of all processing times, the result is the area of the rectangle that has height *m*, width *C* and area *C.m*. Dividing both sides of Equation 3 by $\sum_{j,i} p_{ji}$ we get:

$$1 + \frac{\sum_i l_i}{\sum_{j,i} p_{ji}} = \frac{C.m}{\sum_{j,i} p_{ji}} \tag{4}$$

The left-hand-side of Equation 4 is C' given by Equation 2, thus yielding:

$$C' = \frac{C.m}{\sum_{j,i} p_{ji}} \tag{5}$$

[1] They are available at http://www.mirshekarian.me and can be used freely for research purposes.

This means that for a given problem instance with known m and $\sum_{j,i} p_{ji}$, knowing the efficiency of its optimal solution gives us the makespan of the optimal solution. This is valuable, because if we find a way to predict optimal schedule efficiency without calculating the optimal schedule, we will be able to predict the optimal makespan as well. As mentioned before, this can help us design better heuristics, or have a better insight into the relationship of problem features and optimal makespan.

The way we approached this problem was to first design and learn binary classifiers to get a sense of how to predict optimal scheduling efficiency C'_{min} and then use the obtained knowledge and features to actually predict a numerical value for it. The first phase which is described in this paper involves classifying JSSP instances sampled from a specific class (with specific distributions and size) as having a C'_{min} higher or lower than class average. Support Vector Machines (SVM) have been shown to be very powerful for designing classifiers, and we use them along with the k-Nearest-Neighbors (k-NN) method for comparison. The k-NN method might give inferior results to a well-tuned SVM, but since it does not have any training, it is useful for special cases where training time is critical (e.g. when training data continuously changes and it is not possible to train the model over and over again). On the other hand, if prediction time is more important, SVMs are usually preferred. We used SVMLight [13] and MATLAB as machine learning tools, and experimented with both polynomial and Gaussian kernels with a simple grid search to fine-tune kernel parameters for every setting. To assess feature importance, we used both filter and wrapper methods. In a filter method, the correlation of each feature with the label is measured using conventional correlation measures such as the Pearson Correlation Coefficient (PCC), the T-test for correlation and the Signal-To-Noise ratio (SNR). In a wrapper method, classifiers are trained for each feature or set of features, and the classification performance is used to rank the features.

Features

Selecting the right set of features is a central task in supervised machine learning. Feature engineering is an iterative process that usually involves trial and error and requires basic knowledge about the problem. We developed a preliminary set of 90 features, as listed in Table 2, and experimented with them using the methods explained in previous section. In Table 2, OSMM refers to the total number of machines that are not used in the *i*th operation of all jobs, OSRM refers to the total number of machines that are used by more than one job in the *i*th operation, and OSRMA refers to the OSRM which is amplified by repetition. So if machine 1 is used by 4 jobs at operation slot *i*, it will count as 3 for OSRM, but it will count as 6 for OSRMA, because every repetition will get one extra point than the previous repetition (hence 1+2+3=6). A sample problem is given in Table 1, and the corresponding feature values are given in Table 2. The P and M matrices contain the processing time p_{ji} and machine allocations m_{ji} of each operation ji of job j.

	P					*M*				
J1	96	70	6	58	86	3	1	2	5	4
J2	37	26	23	14	34	3	2	1	4	5
J3	21	83	29	66	25	3	2	5	4	1
J4	18	73	28	29	89	5	2	3	1	4
J5	87	41	32	77	77	5	2	3	4	1

Table 1: Sample problem with 5 jobs and 5 machines

Results and Discussion

Two classes of instances were considered for this paper, one with 10 jobs and 10 machines, and the other with 15 jobs and 15 machines. 5000 random instances were generated with uniform distribution of processing times (ranging from 1 to 100) for each class. Machines were also assigned to jobs randomly, with uniform distribution. All of the instances were solved with IBM ILOG OPL to get their optimal makespan, and labels for supervised learning were calculated based on the difference of the C'_{min} of an instance and the average C'_{min} of the class, which was found to be 1.6330 and 1.5969 for the 10J_10M and 15J_15M classes respectively. 20% of the data was set aside as test data, and 10-fold cross-validation on the of the data was used to fine-tune the parameters. The results are summarized in Table 3.

Domain	**ID**	**Description**	**For Sample Problem**
Instance	1-3	Number of jobs (n), number of machines (m) and total processing time $\sum_{j,i} p_{ji}$	5, 5, 1225
Operations	4-8	Mean, median, standard deviation, minimum and maximum of Operation Processing Times ($\text{OPT}_{ji} \equiv p_{ji}$)	49, 37, 27.8, 6, 96
Jobs	9-13	Mean, median, standard deviation, minimum and maximum of the Job Processing Times ($\text{JPT}_j \equiv \sum_i p_{ji}$)	245, 237, 67.3, 134, 316
	14-17	Median, standard deviation, minimum and maximum of JPT, divided by the mean of JPT	0.97, 0.27, 0.55, 1.29
	18-22	Mean, median, standard deviation, minimum and maximum of JPT, divided by the mean of OPT	5, 4.84, 1.37, 2.73, 6.44
Machines	23-27	Mean, median, standard deviation, minimum and maximum of the Machine Processing Times ($\text{MPT}_i \equiv \sum_{j,k\|M_{jk}=i} p_{jk}$)	245, 226, 43.8, 214, 332
	28-31	Median, standard deviation, minimum and maximum of MPT, divided by the mean of MPT	0.92, 0.18, 0.87, 1.35
	32-36	Mean, median, standard deviation, minimum and maximum of MPT, divided by the mean of OPT	5, 4.61, 8.94, 4.37, 6.77
Operation Slots	37	Standard deviation of the Operation Slot Processing Times ($\text{OSPT} \equiv \sum_j p_{ji}$)	67.80
	38	Standard deviation of OSPT divided by the mean of OSPT	0.28
	39-43	Mean, median, standard deviation, minimum and maximum of the Operation Slot Missing Machines* (OSMM)	2.2, 2, 0.75, 1, 3
	44-47	Median, standard deviation, minimum and maximum of OSMM, divided by the mean of OSMM	0.91, 0.34, 0.45, 1.36
	48-52	Mean, median, standard deviation, minimum and maximum of OSMM, divided by the number of machines (m)	0.44, 0.40, 0.15, 0.20, 0.60
	53-57	Mean, median, standard deviation, minimum and maximum of the Operation Slot Repeated Machines* (OSRM)	2.2, 2, 0.75, 1, 3
	58-61	Median, standard deviation, minimum and maximum of OSRM, divided by the mean of OSRM	0.91, 0.34, 0.45, 1.36
	62-66	Mean, median, standard deviation, minimum and maximum of OSRM, divided by the number of machines (m)	0.44, 0.40, 0.15, 0.20, 0.60
	67-71	Mean, median, standard deviation, minimum and maximum of the OSRM Amplified* (OSRMA)	3.2, 3, 1.7, 1, 6
	72-75	Median, standard deviation, minimum and maximum of OSRMA, divided by the mean of OSRMA	0.94, 0.54, 0.31, 1.87
	76-80	Mean, median, standard deviation, minimum and maximum of OSRMA, divided by the number of machines (m)	0.64, 0.60, 0.34, 0.20, 1.2
Heuristics	81-82	C and C' by the Shortest Processing Time (SPT) heuristic	532, 2.171
	83-84	C and C' by the Longest Processing Time (LPT) heuristic	547, 2.233
	85-86	C and C' by the Least Work Remaining (LWRM) heuristic	624, 2.547
	87-88	C and C' by the Most Work Remaining (MWRM) heuristic	552, 2.253
	89-90	C and C' by the Random (FIFO_MWRM) heuristic	518, 2.114

Table 2: Features and their domain and description; asterisk indicates extra description in text

			10-fold Cross-Validation			*Test*		
	Method	**Parameters**	**Acc.**	**Prec.**	**Rec.**	**Acc.**	**Prec.**	**Rec.**
10J_10M	SVM	Gaussian Kernel, γ=0.05, C=1.0	76.00	75.76	73.01	74.60	75.45	64.75
	k-NN	k=47	73.05	75.12	64.95	71.90	75.05	62.16
15J_15M	SVM	Gaussian Kernel, γ=0.05, C=1.0	72.96	72.03	68.22	74.40	72.94	69.74
	k-NN	k=47	68.24	69.25	56.85	68.80	70.80	56.73

Table 3: Accuracy, precision and recall of cross-validation and test for the two classes

It can be observed that the model has performed with moderate success with the set of preliminary features that was used. It is quite interesting to note that just by using some simple features of a problem instance, features that can be extracted relatively quickly for even a large instance, we are able to predict its efficiency in terms of machine idle time, with more than 75% accuracy. However, this performance seems to have a declining trend when problem size increases, and this issue needs to be addressed. We believe that better temporal features are key in capturing the increased complexity of bigger instances. Specially-designed quick heuristics can be a good candidate.

As for the importance of features, the C' obtained by the five heuristics were ranked the highest using both the filter and wrapper methods. It is to be expected because these are the most elaborate features, and they are the only ones that take into account the time dimension. From the non-temporal features, the standard deviation of JPT divided by mean JPT and mean OPT (features 15 and 20) came first and second, while the mean of OSRMA (feature 67) came third. This is intuitively sensible, because if the standard deviation of an instance is high, there are jobs that need much more processing than others, and these jobs dictate the makespan, keeping many machines idle waiting for them to be finished.

Conclusion and Future Work

We think that the results obtained using our preliminary set of features are promising. We are in search of more temporal and non-temporal features to increase the accuracy to at least 80% and make it stable with problem size. It is also important to look into the kind of mistakes that our classifiers make, and try to see if important aspects are being overlooked when designing new features. Another interesting avenue of further investigation is the use of the knowledge and features obtained in this paper, in designing or choosing automatic dispatching rules. We believe that machine learning is the ultimate tool for designing dispatching rules and the search should be for the right set of features.

References

[1] J.K. Lenstra, and A.H.G. Rinnooy Kan. Computational Complexity of Discrete Optimization Problems. *Annals of Discrete Mathematics*, Elsevier, Vol. 4, pp. 121-140, 1979.

[2] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. *Handbooks in Op. Research and Management Science*, Vol 4, 1993.

[3] J.K., Lenstra, A.H.G., Rinnooy Kan, and P. Brucker. Complexity of Machine Scheduling Problems. *Annals of Discrete Mathematics*, Elsevier, Vol. 1, pp. 343-362, 1977.

[4] M. R. Garey, D. S. Johnson, and Ravi Sethi. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, Vol. 1, No. 2, pp. 117-129, May 1976.

[5] T. Gonzalez, and S. Sahni. Flowshop and Jobshop Schedules: Complexity and Approximation. *Operations Research*, Vol. 26, No. 1, pp. 36-52, 1978.

[6] M.L. Pinedo. Scheduling: Theory, Algorithms and Systems. *American Soc. of Mech. Engineers*, 2008.

[7] A.S. Jain, and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, Vol. 113, pp. 390–434, 1999.

[8] J. Błażewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Op. Research*, Vol. 93, No. 1, pp. 1-33, Aug. 1996.

[9] H. Aytug, S. Bhattacharyya, G. J. Koehler, and J. L. Snowdon. Review of machine learning in scheduling. *IEEE Transactions on Engineering Management*, Vol. 41, pp. 165–171, 1994,

[10] J.H. Blackstone, D.T. Phillips, and G.L. Hogg. A state-of-the-art survey of dispatching rules for manufacturing job shop operations, *Int. Journal of Prod. Research*, Vol. 20, pp. 27-45, 1982.

[11] A. Thesen, and L. Lei. An expert system for scheduling robot in a flexible electroplating system with dynamically changing work loads. *Flexible Manufacturing Systems: Operation Research Models and Applications*, Elsevier, pp. 555–566, 1986.

[12] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, Vol. 64, No. 2, pp. 278-285, 1993.

[13] E. Demirkol, S. Mehta, and R. Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, Vol. 109, No. 1, pp. 137-141, Aug. 1998.

[14] T. Joachims. Making large-Scale SVM Learning Practical. *Advances in Kernel Methods - Support Vector Learning*, MIT-Press, pp. 169-184, 1999.