Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Cell Design and Routing of Jobs in a Multisite Make-to-Order Enterprise

Manoj Gupta, R. P. Jagadeesh Chandra Bose, and **Partha Dutta**
Xerox Research Center India
Bangalore 560 103

Abstract

Make-to-order is a production process where the businesses build the product only after an order from the customer is received. A large enterprise may have many such "make-to-order" shops distributed geographically. The cost and time for executing a job in each of these shops may vary. Therefore, it is important for a multisite enterprise to judiciously decide on where to process the jobs. Ideally, an enterprise would like to minimize the cost (or maximize the profit) while meeting the deadlines and at the same time maximize the utilization of the shops. The time to execute jobs can vary based on how the shops are laid out (the design of shops) and the decision of how jobs are routed (among the various shops). Predicting (or estimating) the likely turnaround time (and cost) for various jobs across the different shops enables the routing decision process. In this paper, we address the two important problems of (i) cell-design and (ii) turnaround time prediction and routing of jobs across various shops. We propose (i) a novel approach based on graph partitioning and set cover heuristic to generate a set of cell designs for a shop, (ii) a framework based on machine learning techniques to predict the turnaround time of jobs across various shops, and (iii) a routing algorithm based on dynamic programming and local search heuristic to route jobs such that the overall profit is maximized. We present results of applying the proposed approaches on real-life datasets from a multisite print shop enterprise.

Introduction

With the increase in the demand for customized products, Make-to-Order (MTO) processes are becoming prevalent in multiple industries. Make-to-order (sometimes, also called build-to-order) is a production process where the businesses build the product only after an order from the customer is received (Hendry and Kingsman 1989). MTO allows order specific customization of the product at the cost of higher turn around time (TAT) to deliver the product, and it is prevalent in industries that require a high degree of customization, such as apparels, automobile parts, and print shops. In a MTO shop jobs arrive over time, and each job is primarily defined by its *arrival time*, *due time (deadline)*, the *functions* that need to be performed, and the quantity of required resources for each function. The shop has a set of *machines* and *operators* that cater to these jobs, where the characteristics of each machine are mainly defined by the functions it can perform, its processing speed and setup time. Given that MTO processes typically have significant turnaround time, one of the primary objectives of the shop is to *minimize* the number of late jobs (jobs that are completed after their deadline) while minimizing the cost of executing the job. This work address the problem of reducing late jobs when batches of jobs arrive at an enterprise with multiple MTO shops to serve the jobs, also called multi-site enterprise.

The problem of reducing late jobs in the above setting can be addressed in three important ways. First, designing the individual shops in a way such that the completion time for common jobs are reduced; second, intelligently routing a job to a shop that can complete the job within its deadline; and finally, improving the internal operation of each shop. The third approach has been extensively studied in earlier scheduling literature (Rai et al. 2009; Ebadian et al. 2009). Our work focuses on the first two approaches for improving the performance of multi-site MTOs.

First, we investigate how can we better design a shop to reduce the late jobs. The shop layout, i.e., how machines are positioned in the shop, has a significant influence on the processing time of jobs because delay is incurred to move jobs from one machine to other. One of the important factors in processing time of jobs is *inter-cellular movement*, the movement of jobs between machines that are placed in different *cells* (rooms), as significant amount of time and effort has to be spent for moving the jobs between cells. Given a typical set of jobs that are processed in a shop, the set of machines available in the shop, and the number of cells K, the *cell design problem* refers to the distribution of machines into K cells such that the number of late jobs and inter-cellular movement are minimized.

Next, we consider the routing problem. A large enterprise may have multiple MTO shops distributed geographically (i.e., a multi-site enterprise). Such an enterprise typically has a centralized mechanism to accepts jobs and jobs can be processed across any of the sites. The cost and time for executing a job in each of these shops may vary. Therefore, it is important for a multisite enterprise to judiciously decide on where to process the jobs. Ideally, an enterprise would like to minimize the cost (or maximize the profit) while meeting the deadlines and at the same time maximize the utilization of

the shops. The decision of how jobs are to be routed (among the various shops) has a significant influence on this objective. Predicting (or estimating) the likely turnaround time (and cost) for various jobs across the different shops enables the routing decision process.

In this paper, we address the two important problems mentioned above, viz., (i) the cell-design problem, (ii) turnaround time prediction and routing of jobs in a multi-site enterprise. Specifically, we make the following three contributions:

1. Introduce the cell design problem, and an algorithm for it based on graph partitioning and set cover heuristics to generate a set of cell designs for a shop,
2. a framework based on machine learning techniques to predict the turnaround time of jobs across various shops, and
3. a routing algorithm based on dynamic programming and local search heuristic to route jobs such that the overall cost is minimized (or overall profit is maximized).

Background

Although the aspects presented in this paper can be applied in any make-to-order shop scenario, we use a 'print shop' as an example make-to-order shop. Let $\mathcal{J}$ be the set of jobs to be processed in a shop. Each job, $J \in \mathcal{J}$ is defined by certain characteristics such as: the *arrival time* of the job, the *expected due time* (deadline) of the job, a list of *functions* that needs to be performed/executed for its processing, and the *quantity/units* required for each function.

Table 1 depicts a few example jobs and the functions required for the jobs to be completed. For example, job J_1 arrived on Sep 26th, 2015 at 2:32 pm and its deadline is Sep 29th, 2015, 11:00 am (i.e., the job has to be completed before this time). This job requires the execution of functions `BWPrinting8x11`, `CoverBind`, `Packaging`, and `UPSLabeling` in that sequence with quantities 80,1,1, and 1 respectively, i.e., 80 sheets of `BWPrinting8x11`, 1 unit each of `CoverBind`, `Packaging`, and `UPSLabeling` need to be performed. The job is routed through machines that can serve these functions. The *turnaround time* (TAT) of a job is defined to be the time difference between the actual completion time and the arrival job of the job.

Let F_J denote the sequence of functions that need to be processed for the job J and let $\mathcal{F}^{\mathcal{J}}$ denote the set of all functions in $\mathcal{J}$, i.e., $\mathcal{F}^{\mathcal{J}} = \cup_{J \in \mathcal{J}} F_J$

Each shop has a set of machines that can process functions required for a job. Each machine serves a specific set of functions and a job might have to be served by multiple machines to cater to all of its required functions. Table 2 depicts an example of machines in a shop and the functions that they support. A machine can serve more than one function and there may be many machines that can serve a particular function. Let $\mathcal{M}$ be the set of machines in a shop. Each machine $M \in \mathcal{M}$ serves a set of functions $F_M \subseteq \mathcal{F}^{\mathcal{J}}$. Let $\mathcal{F}^{\mathcal{M}}$ denote the set of collections of functions supported by all machines in the shop, i.e., $\mathcal{F}^{\mathcal{M}} = \bigcup_{M \in \mathcal{M}} \{F_M\}$.

Related Work

Automated cell design has been an active area of research (Rai et al. 2006; Rai 2011; Jacobs 2012). However, those studies focussed only on partitioning of functions into cells and not machines into cells. The proposed approach in this paper partitions machines into cells and also generates a family of cell designs with varying trade-off between number of late jobs and inter-cellular movement. Furthermore, our approach uses simulation to validate generated cell designs and to improvize them iteratively.

Scheduling jobs within a print shop has been studied as one of the job shop scheduling problems. For instance, in (Rai et al. 2009), the authors present scheduling algorithms to reduce cost and turnaround time in a single print shop as part of a larger print shop productivity improvement solution. A key topic in our work is routing jobs to multiple print shops, without altering the scheduling within each print shop. Some recent work (Zhou, Rai, and Do 2011; Kulkarni and Manohar 2015; Paul, Muniyappa, and Manohar 2014) have considered routing jobs to multiple print shops, with the aim of reducing total completion time (sum of turnaround time for a job within a shop, and the transportation time) and cost. However, these previous papers rely on an elaborate simulation system for a print shop for estimating the completion time of a job, whereas we consider a machine learning based approach using historical data to predict the turnaround time of jobs. A data-driven approach has the advantage of being more robust against different (and possibly, unknown) intra-print shop scheduling algorithms, and unreliability of machines and human operators. Moreover, unlike previous work, our routing algorithm is based on an optimal dynamic programming solution to an idealized routing problem, which is subsequently modified through local search to arrive at a good solution for the routing problem in a practical setting (where scheduler may be unknown, and machines and operators may be unreliable).

Although prediction of waiting times in queueing theory is well studied, there is hardly any literature on predicting turnaround times, especially in the print shop domain. (Chen 2003) presents an approach using fuzzy backpropogation to predict the output times in a wafer fabrication scenario. However, the features that are relevant for a print shop are different from that of the wafer fab. Given domain knowledge about the processes, (Van der Aalst, Schonenberg, and Song 2011) presents an approach to predict the turnaround time/completion time of a process instance using process mining. However, the approach presented in this work does not require the knowledge of any process models and relies as a black box approach.

Knapsack problems (KP) have been extensively used for modeling job assignment or scheduling over multiple processors. In particular, three variants of knapsack problem, multiple, multi-dimensional, and multiple choice, have been used to model multi-processor scheduling problems under various constraints (Kellerer, Pferschy, and Pisinger 2004; Martello and Toth 1990). We model the idealized multi-site routing problem as a multiple dimensional multiple knapsack problem (MDMK). Here, each print shop corresponds to a knapsack, and within each print shop there are capac-

Table 1: Example jobs with their characteristics

Jobs	Arrival Time	Due Time	Function Sequence	Quantity
J_1	09-26-2015 02:32:00 PM	09-29-2015 11:00:00 AM	<BWPrinting8x11, CoverBind, Packaging, UPSLabeling>	<80, 1, 1, 1>
J_2	09-29-2014 05:56:00 AM	09-30-2014 02:00:00 PM	<BWPrinting8x11, ColorPrinting8x11, ThreeHoleDrilling, Packaging, UPSLabeling>	< 45, 23, 1, 1, 1>
J_3	09-29-2014 01:19:00 PM	10-01-2014 11:19:00 AM	<BWPrinting8x11, Punch, CoilBind, Packaging, UPSLabeling>	< 36, 1, 1, 1, 1>

Table 2: Example machines and the functions that they support.

Machine	Functions
BWPrinter1	BWPrinting8x11, BWPrinting8x14, BWPrinting11x17
BWPrinter2	BWPrinting8x11, BWPrinting8x14
ColorPrinter1	ColorPrinting8x11, ColorPrinting8x14, ColorPrinting11x17
CoverBind1,	CoverBind
CoilBind1	CoilBind
HoleDriller1	TwoHoleDrilling, ThreeHoleDrilling, FiveHoleDrilling
Punch1	Punch
Packaging1	Packaging

ity constraints for each function, where each function corresponds to a dimension of the knapsack. Although there has been significant work on studying multiple KP, multi-dimensional KP, and multiple choice KP, the more general MDMK problems have been much less studied. In (Song, Zhang, and Fang 2008), the authors use a MDMK problem to model spectrum allocation in cognitive radio networks. However, the problem presented in (Song, Zhang, and Fang 2008) neither considers different profits for different job and shop pairs, nor different capacity values for different function and shop pairs.

Cell Design

As discussed in the introduction, the turnaround time of a job is influenced by the processing capability of the machine (speed) to which the job has been routed to and the time spent in moving the job from one machine to the other. Typically in shops, there are many rooms (*cells*) in which the machines are to be placed. Each cell has a limitation on the number of machines, (D), it can accommodate, e.g., there can be at most $D = 4$ machines in a cell. An improper assignment of machines to cells results in frequent movement of jobs between cells impacting the turnaround time and thereby lateness of jobs. This movement of jobs between cells is referred to as *inter-cellular movement*. In this section, we address the *cell design* problem, i.e., the problem of assigning machines to various cells in a shop, such that **the number of late jobs and inter-cellular movement are minimized**. Given $\mathcal{M}$ machines in a shop and K cells, the problem of cell design is to assign the machines $\mathcal{M}$ to K cells such that no cell has more than D machines.

The cell design problem in general is NP-Hard. We present a novel approach based on graph partitioning and set cover heuristic to generate a set of cell designs with varying number of late jobs and inter-cellular movement and evaluate them using a widely used commercial printshop simulator (Rai et al. 2009; Zhou, Rai, and Do 2011). Our method can be broadly divided into two steps (i) generate an initial cell design and (ii) improvize generated cell design, which are explained in detail below.

Generate Initial Cell Design

Partitioning Functions to Cells To partition machines $\mathcal{M}$ into K cells, we first partition functions $\mathcal{F}^{\mathcal{J}}$ into K cells. We build a weighted graph $G = (\mathcal{F}^{\mathcal{J}}, E, W)$ where the set of nodes correspond to the set of functions $\mathcal{F}^{\mathcal{J}}$ and the edges correspond to the sequences in which the functions are to be processed in the job list $\mathcal{J}$. There exists an edge $e_{ij} \in E$ between nodes $f_i, f_j \in \mathcal{F}^{\mathcal{J}}$ if the functions f_i and f_j are to be executed in sequence in any of the jobs $J \in \mathcal{J}$. In other words $< f_i, f_j > \in F_J$, for some $J \in \mathcal{J}$ (see Figure 1). The weight w_{ij} of the edge, $e_{ij} = (f_i, f_j)$, is set to the cumulative frequency of executing this function sequence across all jobs in $\mathcal{J}$.

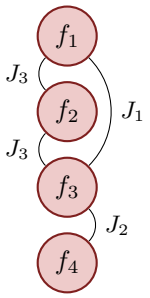


Figure 1: A graph showing relations between jobs and functions. To completed job J_1, function f_1 and f_3 are required (in the same order), so there is an edge between f_1 to f_3.

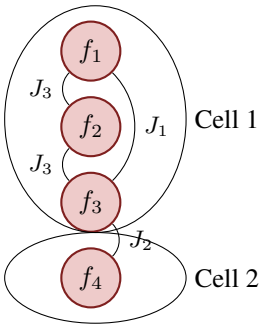


Figure 2: The output of graph partitioning. Here the number of machines = $|\mathcal{M}|$ = 6 and $D = 3$. So the number of partitions required is two. There is only one edge moving from Cell 1 to Cell 2 (edge corresponding to job J_2)

We then partition this graph G into K cells such that the

weighted sum of edges crossing over cells is minimized[1]. Intuitively, the crossing over of edges signify the movement of jobs from one cell to the other. By modelling the cell design problem as a graph partitioning problem, we addressed one of our objectives, that of minimizing the *inter cellular movement*. The graph partitioning results in K partitions where each partition correspond to a cell. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_K\}$ be the K cells. Each cell $C \in \mathcal{C}$ contains a set of functions $F_C \subseteq \mathcal{F}^{\mathcal{J}}$. For example, see Figure 2.

Mapping Machines to Partitioned Functions So far we have partitioned the functions $\mathcal{F}^{\mathcal{J}}$ required to be performed for the jobs. However, our cell design problem should address the partitioning of machines to cells. *We model this problem of assigning machines to cells as a variant of the classical set cover problem.*

Recall that each machine $M \in \mathcal{M}$ can perform a specific set of functions, F_M and $\mathcal{F}^{\mathcal{M}}$ denotes the set of collections of functions supported by all machines in the shop. We need to cover all the functions, $\mathcal{F}^{\mathcal{J}}$, in the job list. This is the *classical set cover problem* where we have a universe of elements (in this case, $\mathcal{F}^{\mathcal{J}}$). Also, there is a collection of sets $\mathcal{F}^{\mathcal{M}}$, such that each set $F_M \in \mathcal{F}^{\mathcal{M}}$ represents a subset of $\mathcal{F}^{\mathcal{J}}$. The aim of the set cover algorithm is to find a set of machines for each cell that can cover all the functions in that cell.

We propose an iterative greedy set cover algorithm that can achieve the above objective (see Algorithm 1).

while *there exists an uncovered function in some cell and there exists some unassigned machine* **do**
  Let M be an unassigned machine that can perform maximum number of uncovered functions in some cell, say C
  if *number of machines assigned to C is less than D* **then**
    Assign M to cell C
  end
end

Algorithm 1: A set cover algorithm for assigning machines to cells

In each iteration we find a machine that can perform maximum number of functions in some cell subject to the condition that the number of machines already assigned to this cell is less than D. This is an important condition because we do not want the number of machines in any cell to be greater than D. We assign this machine to the chosen cell. If we exhaust all the functions or all the machines, then we stop. There are nice theoretical guarantees on the performance of the set cover algorithm though we do not delve upon it in the current paper. Note that it may be the case that we may not cover every function in a cell but the maximum number of machines that could be assigned to a cell is reached. The uncovered functions will be taken care of by machines assigned to some other cell.

[1]Graph partitioning tools such as (Karypis and Kumar 1995) can be used for this purpose.

Dealing with Unassigned Machines There could be a scenario where we have covered all the functions but have not assigned all the machines. This means that some cells have less than D machines. Even though these cells have less than D machines, all their functions are covered. We now assign these unassigned machines as described in Algorithm 2.

foreach *cell $C \in \mathcal{C}$ where the number of assigned machines is less than D* **do**
  while *there exists some unassigned machine* **do**
    Let M be an unassigned machine that can perform maximum number of functions in C
    Assign M to cell C and set M to have been assigned
  end
end

Algorithm 2: Assigning unassigned machines

Note the critical difference between this algorithm and the classical set cover algorithm. In this algorithm, we do not want to cover all the functions as they are already covered by the set cover algorithm. So, we do the next best thing, for each machine M we choose a cell in which it can perform maximum number of functions subject to the condition that the number of machines in this cell is less than D. This cell already has a machine M' that can perform the functions that M can perform. But by assigning M to this cell, we increase the number of machines that can perform those functions. At the end of this, each cell $C \in \mathcal{C}$ contains a set of machines $M_C \subseteq \mathcal{M}$.

Improving Cell Design using Shop Simulation

The cell design method described so far focusses only on the inter-cellular movement and did not consider the other crucial objective of the number of late jobs. In this section, we address that objective through simulations of the shop under consideration. For performing simulations, we use a commercial printshop simulator (Rai et al. 2009; Zhou, Rai, and Do 2011).

For a given cell design $\mathcal{C}$ and a job list $\mathcal{J}$, let LATEJOBS$(\mathcal{J}, \mathcal{C})$ denote the number of late jobs and INTER-CELLULARMOVEMENT$(\mathcal{J}, \mathcal{C})$ denote the total inter-cellular movement of jobs. Let UTIL$(M, \mathcal{C})$ denote the utilization of machine M. These metrics can be obtained by the simulator upon simulating a job list on a cell design.

Tradeoff between late jobs and inter-cellular movement Achieving both the objectives of cell design (i.e., minimizing the number of late jobs and inter-cellular movement of jobs) at the same time may be difficult. For example, consider the following cell design (with at most 2 machines per cell, i.e., D = 2), $\mathcal{C} = \{C_1, C_2\}$ where C_1={BWPrinter1, Binder1}, and C_2 ={BWPrinter2, Binder2}. Assume that the simulation outputs the following metrics on a job list of 30 jobs: LATEJOBS$(\mathcal{J}, \mathcal{C}) = 20$, INTER-CELLULARMOVEMENT$(\mathcal{J}, \mathcal{C}) = 0$ and the utilization of various machines to be UTIL(BWPrinter1, $\mathcal{C}$) = 70, UTIL(Binder1, $\mathcal{C}$)=20, UTIL(BWPrinter2, $\mathcal{C}$)=10, UTIL(Binder2, $\mathcal{C}$)=10.

The number of late jobs in this scenario is quite high even though the inter cellular movement is 0. One justification for this scenario could be that most of the jobs are directed to cell C_1 by the simulator, which is reflected in the high utilization of machine BWPrinter1 (utilization= 70). As a remedy, we can move a similar under-utilized machine, i.e., BWPrinter2 in cell C_2 to C_1. This transfer increases the number of machines in C_1 to 3. Since $D = 2$, we cannot have more than 2 machines in any cell. So, we find any under-utilized machine in cell C_1, i.e., Binder1 in this case, and move it to C_2. The new cell design $\hat{\mathcal{C}}$ is as follows: $\hat{C_1}$={BWPrinter1, BWPrinter2}, $\hat{C_2}$={Binder1, Binder2}. Again, we run the simulation on the cell design $\hat{\mathcal{C}}$. In this cell design all the jobs requiring BWPrinting has to go to $\hat{C_1}$ for BWPrinting. However, there are two machines in $\hat{C_1}$ that can take much more load. So the simulation results are as follows: LATEJOBS$(\mathcal{J}, \hat{\mathcal{C}}) = 6$, INTER-CELLULARMOVEMENT$(\mathcal{J}, \hat{\mathcal{C}}) = 10$ and the utilization of various machines to be UTIL(BWPrinter1, $\hat{\mathcal{C}}$) = 40, UTIL(Binder1, $\hat{\mathcal{C}}$)=20, UTIL(BWPrinter2, $\hat{\mathcal{C}}$)=30, UTIL(Binder2, $\hat{\mathcal{C}}$)=10.

We can see that the number of late jobs decreases from 20 to 6. However, the inter-cellular movement of jobs increases from 0 to 10, i.e., 10 jobs need to be moved from BWPrinting to Binding (all the other jobs needed either printing or binding). The above example presents a dilemma to the cell design problem. Our objective is to minimize both the number of late jobs and the inter-cellular movement . However, this objective may not be practical always and sometimes we may have to sacrifice in one of the parameters (in our case we are always willing to sacrifice the inter-cellular movement).

Heuristic to balance late jobs Our initial cell design used graph partitioning technique, that minimizes the inter-cellular movement. But as the above example suggested, this may not be the best policy always. Our initial cell design may lead to higher number of late jobs(as reported by the simulator). Next, we design a heuristic, broadly based on the famous heuristic of Kerninghan and Lin (Kernighan and Lin 1970), to improve it. Algorithm SWAPANDSIMULATE takes a real parameter τ, whose value can be between 0 and 100 (typically 40 in our experiments). We want the utilization value of each machine M to be less than τ for each improved cell design created by SWAPANDSIMULATE.

We now describe this heuristic in detail. We introduce another parameter α (initially set to a high utilization value of 90). In the first iteration of our algorithm, we find a machine M (in cell C_i) such that the utilization of M is greater than α. If such a machine is found, then we search if there exists another machine M'(equivalent to M in another cell, C_j) with low utilization (UTIL$(M', \mathcal{C}) < \tau$). If we find such a machine M', then we assert if the total number of machines in C_i is less than D. If yes, then we can conveniently move machine M' from cell C_j to cell C_i. If not, then we search for an under-utilized machine M'' in cell C_i (with UTIL$(M'', \mathcal{C}) < \tau$). If yes, then we swap machines M' and M'' from cells C_i and C_j. This ensures that the number of machines in C_i is exactly equal to D.

```
C ← initial cell design.
α = 90
while α > τ do
    Run the simulator on the current cell design C.
    if there exists a machine M in cell C_i with
    UTIL(M, C) > α then
        if there exists a machine similar to M, say M' in
        cell C_j, such that UTIL(M', C) < τ then
            if the number of machines in C_i < D then
                Move M' from C_j to C_i
            end
            else if there exists machine in C_i, say M'',
            which is not similar to M and
            UTIL(M'', C) < α then
                Move M' from C_j to C_i
                Move M'' from C_i to C_j
            end
        end
    end
    if the cell design C was changed then
        Let Ĉ be the new cell design after the change
        Run the simulator on the new cell design Ĉ
        if LATEJOBS(J, Ĉ) < LATEJOBS(J, C) then
            C ← Ĉ
        end
        else
            α ← α − 5
        end
    end
    else
        α ← α − 5
    end
end
```

Algorithm 3: SWAPANDSIMULATE(τ)

Let $\hat{\mathcal{C}}$ be the new cell design after the change. The quality of this new cell design is assessed using the simulator. If the number of late jobs in $\hat{\mathcal{C}}$ is less than $\mathcal{C}$, then our new cell design is better. In this case, we adopt $\hat{\mathcal{C}}$ as the current best cell design (by assigning $\mathcal{C} \leftarrow \hat{\mathcal{C}}$). If not, then we either have not been able to swap any machines or even after swapping our new cell design is not able to better the current cell design. This may imply that the bottleneck is not the current machine M but some other machine whose utilization value is not greater than α but is greater than τ. So, we decrease the value of α by a constant amount (say, 5) and iterate.

We present the evaluation of the proposed cell design in the Experiments section. Next, we discuss another important problem of routing jobs over multiple sites. We will see that our local heuristic will be used for this problem as well.

Turnaround Time (TAT) Prediction

In this section, we describe a framework for turnaround time (TAT) prediction for jobs. A TAT prediction model is learned, one for each shop, based on historical data pertaining to that shop. The learned model can then be used to predict the likely TAT for future jobs. The predicted TAT is used

in the routing decision process (described in the next section).

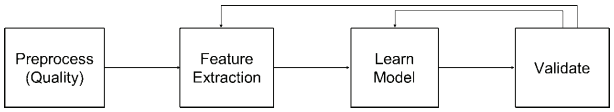


Figure 3: Framework for TAT prediction

Figure 3 depicts the framework for TAT prediction using machine learning. The basic building blocks are explained below:

Preprocess: This corresponds to detection of any quality issues w.r.t the data and identification of outliers, infrequent jobs, etc. For machine learning models to perform well, a statistically significant number of samples need to be present in each of the classes. It could be the case that some job types are very infrequent; such jobs can be ignored.

Feature Extraction: This corresponds to the definition and extraction of features, which are the basis for prediction models. The TAT of a job primarily depends on the:

1. characteristics of the job such as the functions required and their quantities, the available time to print the job, etc.
2. characteristics of the shop such as the stations available for various functions, the processing speed, the operators available for each of the functions, the duration for which the stations/operators are available etc.
3. workload at the shop at the time of arrival of a job. The scheduling of job depends on prior workload (jobs that are still pending to be executed or under execution).

We define various features to enable the building of a TAT prediction model.

Function Quantity: A vector of size equal to the number of distinct functions in all jobs, i.e., $l = |\mathcal{F}^{\mathcal{J}}|$. For each job, an element of the vector corresponds to the number of units of the function corresponding to the element in the job. The function quantity vector of a job J corresponds to $[q_1, q_2, \ldots, q_l]$ where q_i denotes the quantity of function i that needs to be processed.

Workload Vector: A vector of size equal to the number of distinct functions in all jobs. For each job and for each function in the job, an element of the workload vector pertaining to the function corresponds to the cumulative sum of units of the function of all unfinished jobs that have arrived before this job. Let $\mathcal{U}_J$ denote the set of all jobs that have arrived before job J and those that are yet to be processed. Then the workload vector for job J corresponds to $[wl_i, wl_2, \ldots, wl_l]$ where wl_i denotes the workload of function i and is equal to $\sum_{O \in \mathcal{U}_J} q_i^O$ where q_i^O denotes the quantity of function i in job O.

Station Vector: A vector of size equal to the number of distinct functions in all jobs. For each job and for each function in the job, the element of station vector pertaining to the function corresponds to the cumulative number of stations that support the function

Operator Vector: A vector of size equal to the number of distinct functions in all jobs. For each job and for each function in the job, the element of operator vector pertaining to the function corresponds to the cumulative number of operators that are skilled to process the function

Available time: It is basically the amount of time available to execute the job since its arrival. It is defined as the time difference between the due date and arrival date, i.e., $\mathrm{availabletime} = \mathrm{duetime} - \mathrm{arrivaltime}$.

Turnaround time: It is the amount of time taken to finish the job since its arrival and is defined as the time difference between the completion time and arrival time, i.e., $\mathrm{Turnaroundtime} = \mathrm{completiontime} - \mathrm{arrivaltime}$

The turnaround time feature is the output variable of the prediction model.

Variants of available time and turnaround time are also defined considering the working hours (shifts) of the shop. *Shift adjusted available time* is the amount of working time available for the job to be processed. *Shift adjusted turnaround time* is the amount of working time taken to process the job considering only the working hours of the shop. Furthermore, the times (both regular as well as shift adjusted) can be divided into bins of certain size and considered for learning/prediction. For example, if the available time is 20 hours and we consider bins of size 3 hours, then the available time is $\lceil 20/3 \rceil = 7$ bins. By considering bins, our objective of predicting turnaround time boils down to identifying the "bin" in which the job is likely to get finished. Since SLAs are mostly written at processing jobs within a certain duration, a bin based feature is promising to be considered. For each job, the features defined above can be extracted and appended as a single "feature vector" with the TAT based feature as the output feature and fed into a learning algorithm. The set of all such feature vectors for the data set constitutes the data matrix.

Learn Model: The problem of predicting the TAT turns out to be a regression problem. The data matrix defined above can be subjected to various regression learning algorithms and an appropriate learning algorithm can be chosen based on their performance. We have chosen Artificial Neural Networks as the learning algorithm. During the learning phase, the learning algorithm is trained using a k-fold cross validation technique (where k typically is set to 10). In a k-fold cross validation, the model is trained with $(100 - 100/k)\%$ of the training data and $100/k\%$ of test data and this is repeated "k" times with each time having a different set for training and testing.

Validate: The performance of the learning algorithm is evaluated on how well the learned model is able to predict the TAT time. We define the accuracy considering up to m-deviations from the actual TAT (where 'm' is typically set to 2). A predicted output is considered to be correct if the predicted value is within 'm' units from the actual output. For example, if the actual output (TAT) of a job is 3 bins and if the learned model predicted it to be 2, 3, or 4, we consider it to be correct under $1-$deviation.

Once a model is learned, any new job can be fed into the model, which returns its expected TAT as output.

Multisite Routing

In this section, we present an approach for determining how jobs are to be routed across the various shops using the TAT prediction model described above. We assume that jobs arrive in batches and all jobs in a batch have the same deadline. Let l denote the distinct number of functions that need to be processed, i.e., $l = |\mathcal{F}^{\mathcal{J}}|$. Let m be the number of shops in the enterprise and let n be the number of jobs to be routed. Let $W_i = [w_{i1}, w_{i2}, \ldots, w_{ik}, \ldots, w_{il}]$ denote the quantities of various functions for job i. Let $Z_j = [z_{j1}, z_{j2}, \ldots, z_{jk}, \ldots, z_{jl}]$ denote the total processing capacity of various functions in shop j for a certain time interval, i.e., z_{jk} denotes the total processing capacity of function k. Let p_{ij} denote the profit of processing job i at shop j. Let x_{ij} be a variable that denotes whether job i is processed at shop j; $x_{ij} = 1$ if job i is routed at shop j and 0 otherwise. Then, the processing capacity of a shop j for a function k results in the following constraint: $\sum_i \sum_k x_{ij} w_{ik} \leq z_{jk}$. Our aim is then to increase the total profit for routing the jobs. Our problem can be formulated as a linear program.

$$\begin{aligned} \max \quad & \textstyle\sum_i \sum_j x_{ij} p_{ij} \\ \text{s.t.} \quad & \\ & \textstyle\sum_i \sum_k x_{ij} w_{ik} \leq z_{jk} \\ & \\ & x_{ij} = \{0, 1\} \end{aligned}$$

Our problem is a variant of Multiple Dimension Multiple Knapsack Problem (MDMK) (Kellerer, Pferschy, and Pisinger 2004). We now present an algorithm for this problem using dynamic programming approach. We create a multi-dimensional array B as follows: $B[i, A_1, \ldots, , A_m]$ is the profit obtained by routing the first i jobs in shops such that:

- A_j represent the vector $[a_{j1}, a_{j2}, \ldots, a_{jl}]$, signifying the current processing capability at shop j, initially set to Z_j.
- for a shop j, $\sum_i \sum_k x_{ij} w_{ik} \leq a_{jk}$.

Note that not all of the first i jobs may routed; some jobs (say having low profit) may be dropped. As a base case, $B[1, A_1, \ldots, A_j, \ldots, A_m] = p_{1j}$ (job 1 is routed at shop j) if $w_{1k} \leq a_{jk}$ (for all k). Assume that we have already calculated $B[i-1, \cdot, \ldots, \cdot]$. The matrix B can be filled using the following recursion.

$$\begin{aligned} B[i, A_1, \ldots, A_m] = \max\{ & \\ & B[i-1, A_1, \ldots, A_m], \\ & B[i-1, A_1 - W_i, \ldots, A_m] + p_{i1}, \\ & .., \\ & B[i-1, A_1, \ldots, A_m - W_i] + p_{im}\} \end{aligned}$$

The first element above indicates the scenario where the job i cannot be routed to any of the shops (due to unavailable processing capacity) and the later items refer to jobs being routed to one of the m shops. Note that a job is assigned to shop j only if $A_j - W \geq \mathbf{0}$. At the end, the cell $B[n, Z_1, Z_2, \ldots, Z_m]$ denotes the optimal profit of routing n jobs. The above algorithm can be augmented to find the actual routing of jobs (through backtracking).

Lemma 1 *$B[i, A_1, \ldots, A_m]$ is the optimal profit of routing the first i jobs to m shops where shop j has total processing capability A_j.*

Local Improvement over the Initial solution

Our dynamic programming approach might yield suboptimal solutions since it does not take into account the current load at each shop. For example, assume that a shop already has many jobs in its queue, when the current batch arrives for routing. Then, a reasonable approach would be to either not route or route few jobs to this shop. However, the dynamic programming approach does not use this information of current load while allocation of job. We now describe a procedure that, starting from the solution given by dynamic programming, will use our TAT Prediction module to arrive at a local optimum for the routing algorithm.

Given a list of jobs $\hat{\mathcal{J}}$ routed at shop j, we first find the turnaround time for each job. To this end, we use the TAT Prediction algorithm that has the full knowledge of workload at each shop. We now do local improvement on the initial assignment (given by dynamic programming) as follows:

Let $\hat{\mathcal{J}}$ be the job list assigned to shop j.
Run the TATPrediction Algorithm at shop j and job list $\hat{\mathcal{J}}$.
Let $L_{\hat{\mathcal{J}}}$ be the set of late jobs at shop j.
while *true* **do**
  Find a late job i at shop j.
  if *there exists a shop j' such that $z_{j'k} - a_{j'k} \geq w_{ik}$ (for all k), where $a_{j'k}$ is the current processing capacity of function k at shop j' before the current batch arrives* **then**
    Move the job i from j to j'
  end
end

Algorithm 4: A local algorithm to improve routing decisions

Algorithm 4 finds a late job i at shop j and tries to find an alternate shop j' such that all the functions required by job i can be met at this shop.

Experiments and Discussion

In this section, we present and discuss some experimental results.

Cell Design We performed extensive experiments on the cell design method presented in the previous section. For our experiments, we use real data from print shops of a large document processing company with their known set of machines $\mathcal{M}$, the job list $\mathcal{J}$, and a parameter, D, signifying the maximum number of machines in each cell. We present the results of one such experiment where the shop has 223 jobs and about 74 machines. We generated designs with varying number of cells. Figure 4 depicts the influence of number of cells on the late jobs and inter-cellular movement as obtained from the simulator. We can see that the inter-cellular movement increases as the number of cells increases. The number of late jobs tend to decrease for a design with three cells and increases later for four cells. This could be due to

the increased inter-cellular movement. Clearly, from the figure, we can assess the trade-off between number of late jobs and inter-cellular movement and made a judicious choice on cell design (e.g., a choice of 3 cells provides a reasonable trade-off between between two objectives for this data).

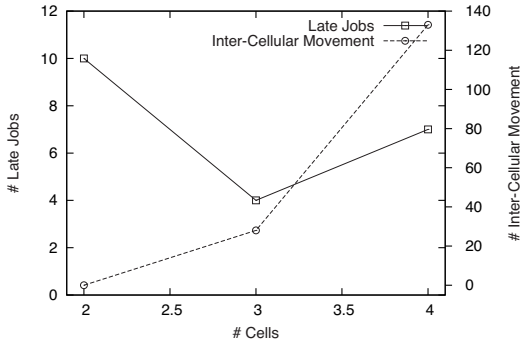


Figure 4: Influence of number of cells on late jobs and inter-cellular movement.

TAT Prediction We have evaluated the TAT prediction approach on several real-data sets obtained from multiple print shops. We present the results of one such data set from a large university print shop, which consisted of 3313 jobs. There were 25 distinct functions in the job list. The shop had 21 stations catering to the 25 functions required for processing the jobs. We have considered shift-adjusted variants of available and turnaround times and a bin-size of 3 hours. We processed the raw data and generated features as discussed earlier (ref. Section TAT Prediction) and learned a artificial neural network based model. Table 3 depicts a summary of accuracy results on this dataset using a 10-fold cross validation. We can see that we are able to achieve an accuracy as high as 99.3% with a deviation of 2 bins.

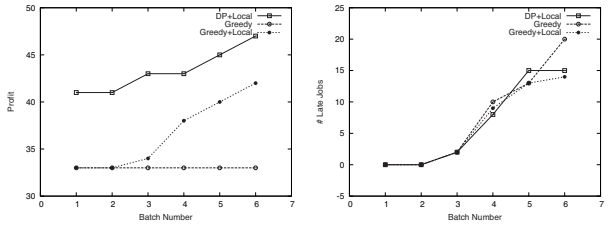
Table 3: Accuracy results of TAT prediction approach

Deviation	#Correctly Classified	Accuracy (%)
0	2600	78.5
1	2953	89.1
2	3292	99.3

Routing Algorithm We evaluate our proposed Dynamic Programming (DP) with local search routing algorithm and compare it with the following variants: (1) Greedy Algorithm: The greedy algorithm puts a job in a shop if the shop can process the job given its current load. (2) Greedy Algorithm + Local Search, which runs the local search algorithm over an initial solution obtained by the Greedy Algorithm.

We considered 32 jobs in a batch requiring 12 functions and three shops for this experiment. In order to study the efficacy of the routing algorithm, we generated 5 additional batches by modifying their deadlines. In each batch, we reduced the available time for each job by 3 hours from its previous batch. So, the 6th batch has jobs whose available time is 15 hours shorter than the first batch. Clearly as the batch number increases, the available times are shorter and we expect the number of late jobs to increase.

For each job, its processing cost is directly proportional to the size of the job and the number of function required to process it. The profit of a jobs is then defined to be a fixed ratio (say, 0.5) of its processing cost.



(a) (b)

Figure 5: Comparison of routing algorithms.

Figure 5 depicts the results of the three routing strategies. We can see that our proposed DP + Local search algorithm maximizes the profit among the variants (Figure 5(a). The profit of Greedy is constant (at 33) as jobs are same in each batch (and Greedy assignment is same in each batch). The local improvement over Greedy is better than Greedy in terms of their profit. As far as the number of late jobs is concerned (Figure 5(b)), we conclude that DP + Local Search is either better or comparable to both Greedy and Greedy+Local Search. As expected, the number of late jobs increases as the batch number increases in all the three algorithms with increasing batch size.

Conclusions and Future Work

In this paper we proposed techniques that enable multi-site make-to-order enterprises process jobs minimizing the number of late jobs and maximizing the profit. The proposed techniques have been shown to be effective on real-life datasets. As future work, we would like to explore other regression techniques for turnaround time prediction. Furthermore, in this work we considered the routing of jobs in a batch assuming that all jobs within a batch have the same deadline. However, in practice, in most applications jobs within a batch can have varying deadlines. We would like to extend the routing algorithm to consider cases where each job can have its own deadline.

References

Chen, T. 2003. A fuzzy back propagation network for output time prediction in a wafer fab. *Applied Soft Computing* 2(3):211–222.

Ebadian, M.; Rabbani, M.; Torabi, S.; and Jolai, F. 2009. Hierarchical production planning and scheduling in make-to-order environments: reaching short and reliable delivery dates. *International Journal of Production Research* 47(20):5761–5789.

Hendry, L. C., and Kingsman, B. 1989. Production planning systems and their applicability to make-to-order companies. *European Journal of Operational Research* 40(1):1–15.

Jacobs, T. 2012. Creating workflows for a job shop, assigning the workflows to cells of devices, and splitting the workflows within complex cells. US Patent 8,259,331.

Karypis, G., and Kumar, V. 1995. Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0.

Kellerer, H.; Pferschy, U.; and Pisinger, D. 2004. *Knapsack problems*. Springer.

Kernighan, B. W., and Lin, S. 1970. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal* 49(2):291–307.

Kulkarni, K., and Manohar, P. 2015. Cost efficient short term capacity planning for MTO enterprises. In *Proceedings of the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation (PADS)*.

Martello, S., and Toth, P. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc.

Paul, R.; Muniyappa, M.; and Manohar, P. 2014. Job routing in multi-site print shop environment. In *Big Data Analytics - Third International Conference, BDA. Proceedings*.

Rai, S.; Godambe, A.; Duke, C.; and Williams, G. 2006. Printshop resource optimization via the use of autonomous cells. US Patent 7,079,266.

Rai, S.; Duke, C. B.; Lowe, V.; Quan-Trotter, C.; and Scheermesser, T. 2009. LDP lean document production - o.r.-enhanced productivity improvements for the printing industry. *Interfaces* 39(1):69–90.

Rai, S. 2011. System and method for assigning print jobs to autonomous cells in a transaction printing environment. US Patent 8,059,292.

Song, Y.; Zhang, C.; and Fang, Y. 2008. Multiple multidimensional knapsack problem and its applications in cognitive radio networks. In *IEEE Military Communications Conference (Milcom)*.

Van der Aalst, W. M.; Schonenberg, M. H.; and Song, M. 2011. Time prediction based on process mining. *Information Systems* 36(2):450–475.

Zhou, R.; Rai, S.; and Do, M. 2011. Scheduling print jobs in lean document production (ldp) toolkit. In *System Demonstrations and Exhibits, Twenty-First International Conference on Automated Planning and Scheduling, ICAPS 2011*.