

A FLUID HEURISTIC FOR MINIMIZING MAKESPAN IN JOB-SHOPS

J. G. Dai*
Gideon Weiss†

January 11, 1999

First Revision: May 7, 2000

Second Revision: July 7, 2000 and March 9, 2001

To appear in *Operations Research*.

Abstract

We describe a simple on-line heuristic for scheduling job-shops. We assume there is a fixed set of routes for the jobs, and many jobs, say N , on each route. The heuristic uses safety stocks and keeps the bottleneck machine busy at almost all times, while the other machines are paced by the bottleneck machine. We perform a probabilistic analysis of the heuristic, under some assumptions on the distributions of the processing times. We show that our heuristic produces makespan which exceeds the optimal makespan by no more than $c \log N$ with a probability which exceeds $1 - 1/N$ for all $N \geq 1$, where c is some constant independent of N .

1 The Job-Shop Scheduling Problem with Fixed Routes

A job-shop consists of machines $i = 1, \dots, I$, and routes $r = 1, \dots, R$. Route r consists of steps (r, k) where $k = 1, \dots, K_r$ indicate the steps along route r , in their required order of execution, and step (r, k) is carried out by machine $\sigma(r, k)$. We let C_i denote the set of steps performed on machine i . In the standard job-shop formulation [23] there is one job on each route, and the objective is to schedule all the jobs so as to minimize the makespan, the earliest time by which all the jobs are completed.

In our formulation of the job shop problem we assume that there are many jobs on each of the routes. In practice, in particular in factories, routes may correspond to various production

*School of Industrial and Systems Engineering and School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332-0205, USA; Research supported in part by NSF grants DMI-9457336 and DMI-9813345, US-Israel BSF grant 9400196; Part of this work was completed while this author was visiting MaPhySto—Centre for Mathematical Physics and Stochastics, funded by a grant from the Danish National Research Foundation.

†Department of Statistics, Haifa University, Mount Carmel, Haifa 31905, Israel; Research supported by US-Israel BSF grant 9400196 and German-Israel GIF grant I-564-246.05/97.

processes, or to various types of products manufactured in the factory. In that case the jobs may correspond to parts or lots and there will indeed be many such jobs for each route.

We are given a set of jobs \mathcal{N} , partitioned by their routes into subsets \mathcal{N}_r , of size $N_r = |\mathcal{N}_r|$. Job $j \in \mathcal{N}_r$ requires steps (r, k) with processing by machines $\sigma(r, k)$, and processing times $X_{r,k}(j)$.

Throughout this paper we consider R as fixed, and let N_1, \dots, N_R grow. In fact we focus on the case of $N_r = n_r N$, and we let N grow. We call N the *multiplicity* of the job-shop problem. Without loss of generality, we can take $n_r = 1$ and thus $N_r = N$, since we can think of n_r jobs on a single route r as single jobs on n_r routes. The more general case where N_r grow independently is discussed briefly in Section 5.

A schedule is an assignment of machines to jobs (one to one, each job can be assigned to one machine at a time, and each machine can be assigned to only one job at a time, with the possibility that jobs or machines are idle) to perform all the steps of each job in the order required by its route, with no preemption (interruption) in the middle of any job step. A schedule results in departure times of the jobs $D(j)$ (for job $j \in \mathcal{N}_r$ this is at the completion of step (r, K_r)). The job-shop scheduling problem is to find a schedule that minimizes the makespan (time of the last job completion), defined as $\max\{D(j) : j \in \mathcal{N}_r, r = 1, \dots, R\}$.

The standard job-shop scheduling problem is NP-hard [15]. It is in fact notoriously hard. While anybody may be able (with considerable effort) to schedule 4 jobs on 3 machines, going to as few as 10 jobs reaches the frontier of expertise in combinatorial optimization. One particular instance of a 10 job, 10 route, 10 machine problem (in which the route of each job passes once through each machine), which was randomly generated as an example in an early scheduling textbook [28], was taken up by the combinatorial optimization community as a challenge and bench mark. Named the 10×10 scheduling problem, it took 12 years of continuous effort to solve [8]. Currently some heuristic methods will reach the optimal solution for this problem, with a value of 930, in about 20 seconds of computing [1, 26], but this is without an optimality proof, and the heuristics do not provide performance guarantees. The optimal solution of job shop problems with 20 jobs on 20 routes, each route consisting of 20 steps, through a set of 20 machines, remains beyond the reach of current methods.

There seems to be nothing in the theory to indicate that multiple jobs which follow the same route make the problem easier than if each job follows a different route. For example, if we look at the 10×10 problem and we formulate a new problem, which we call the N -multiplied problem, with $10N$ jobs, in which there are N exact copies of each of the original 10 jobs, and we compare that with a general problem of 10 machines and $10N$ jobs, each with its own 10 step route, then we are not aware of solution techniques which distinguish between the two. Thus for the purpose of exact optimization, our impression is that the new problem with multiplicity $N = 10$ cannot be solved optimally by current methods. However, our contention is that the approximate solution of problems with high multiplicity is much easier, because of the following argument.

A well known simple lower bound on the makespan can be computed easily through the workloads on the machines. The workload of machine i consists of $T_i = \sum_{(r,k) \in C_i} \sum_{j \in \mathcal{N}_r} X_{r,k}(j)$. Clearly the makespan cannot be less than any of these values. Hence a lower bound on the

makespan is:

$$\text{makespan} \geq T^* = \max_{1 \leq i \leq I} T_i = \max_{1 \leq i \leq I} \sum_{(r,k) \in C_i} \sum_{j \in \mathcal{N}_r} X_{r,k}(j). \quad (1.1)$$

This is called the machine lower bound. For the 10×10 scheduling problem this bound equals 631. Apart from the machine lower bound, there is a job lower bound, given by the length of the longest job, as $\max_{j,r} \sum_{k=1}^{K_r} X_{r,k}(j)$. For the 10×10 problem this is 655. If we now look at the N -multiplied 10×10 problem, its machine bound equals $631N$, while its job bound remains 655. More generally, if we have many jobs on each route, and we assume that none of them is dominant, then the job bound is dominated by the machine bound. Hence we might expect that the optimal solution is quite close to the machine lower bound.

Our heuristic attempts to get close to this machine lower bound. We are motivated by a *fluid approximation* to the job-shop problem. For the approximation we assume that work is composed of homogeneous fluid rather than discrete jobs with various processing times, and we assume that fluid out of buffer (r, k) , which stores the work of step (r, k) , could flow at a rate which is proportional to the machine effort of machine $\sigma(r, k)$. If the full effort of the machine is devoted to buffer (r, k) flow will be at the maximal rate of $1/\bar{m}_{r,k}$, where $\bar{m}_{r,k}$ is the average processing time at step (r, k) among all N_r jobs. We can then make machine i work at a constant fraction T_i/T^* of its processing capacity, and divide its processing capacity between all its buffers so that the fluid in all of them decreases at the same rate. The bottleneck machine ($\arg \max T_i$) would work at full capacity, and all the other machines would work at a fraction of their capacity and keep pace with the bottleneck machine. Under this policy the amount of remaining work for each machine will decrease at a constant rate, and all the machines will finish all their work simultaneously, at T^* . Hence this fluid solution would achieve exactly the machine lower bound (see [36] for details). Our heuristic tries to schedule the jobs so as to imitate the fluid solution. We therefore call it a *fluid heuristic*.

Because our jobs are not fluid, but are discrete jobs with various processing times, and each machine can only work on one job at a time, we cannot follow the fluid solution precisely. To get close to it we use safety stocks of size $O(\log N)$. The exact size of the safety stocks should take into account the values of T_i/T^* , which we call the machine utilizations, and the variability of the processing times on each route. In practice, the identity of the bottleneck machine, the machine utilizations, and the variability of the processing times will be estimated from general information about the processing times, rather than from the exact values of all the $X_{r,k}(j)$. In this paper we simply choose the same safety stocks in all the buffers.

Once we determined the bottleneck machine and the size of the safety stocks, our heuristic operates *on-line* in the following sense: We do not use the processing time values in order to choose some favorable permutation of the jobs, which is the usual off line, time consuming part of scheduling algorithms. Instead we fix the order of the jobs and renumber them, and we perform each step (r, k) for the N jobs of \mathcal{N}_r in that order. We also fix the order of the steps $(r, k) \in C_i$ and let machine i go through a cycle of its steps in that order. At any time t , the heuristic uses only the following information: How many jobs have completed each of the steps, which job step is in process on each working machine, and which machines are idle. When this state changes, i.e., at a completion of a step on some machine, we determine from this state information which of the idle machines starts working again. The predetermined order

of jobs and steps then dictates which jobs and what steps will be processed at these starts. The processing times of the jobs are only needed to determine when job steps are completed, namely $X_{r,k}(j)$ time after they start. Thus the heuristic requires an amount of calculation linear in the problem data, and we need not know the processing times of the job-steps in advance. The pre-determined order of the jobs may be the order in which they arrived in the system, or order according to due dates, or some other criterion, but is independent of the actual processing times.

The heuristic is constructed, described, and studied in Section 2. We give a brief overview of it here. We initiate the schedule by processing some of the jobs partially, i.e., we perform only part of the route of each of these jobs. At the end of this initial period we reach a state where there are $S_{r,k}$ jobs which have completed all their processing prior to step (r, k) , and these form a *safety stock* for job-step (r, k) . We let $j_{r,k}$ denote the index of the job of route r which is the first in the safety stock of step (r, k) , and we refer to the $j_{r,k}$ as *offsets*. This initial part of the schedule is followed by $j' = 1, \dots, \tilde{N}$ complete cycles in which we perform step (r, k) on job $j_{r,k} + j' - 1$, where $\tilde{N} = N - \max_{r,k} j_{r,k} + 1$. The offsets are intended to ensure that in each cycle the various machines are working on different jobs, and can work simultaneously. This allows for a high utilization of the machines. During this middle section of the schedule we intend the bottleneck machine to work continuously. The schedule is completed by partial cycles which empty all the safety stock buffers, and complete all the remaining jobs.

During the \tilde{N} cycles, the order of the jobs and job-steps on each machine is predetermined by the fixed order of the jobs, and by the fixed order of the steps. The actual start times of job-steps is as follows: The bottleneck machine uses early starts throughout (job-steps are started as soon as they and the machine are available), while all the other machines delay the start of each cycle until the bottleneck has started the cycle, and use early start thereafter. Thus the bottleneck machine sets the pace for all the other machines during the middle part of the schedule.

How well such a schedule performs depends on the multiplicity N and on the actual processing times of the jobs. To evaluate the performance of the heuristic we perform a probabilistic analysis. Probabilistic analysis of heuristics for scheduling problems has been used in [14, 11, 34, 30, 19, 9, 22].

The probabilistic analysis of the heuristic assumes a random population of problems, where each instance of the problem is drawn randomly from this population. The assumptions on the population are that processing times are random and independently drawn, and the processing times of jobs at each step, $X_{r,k}(j), j \in \mathcal{N}_r$ are identically distributed with some general distribution $F_{r,k}$. Our only assumption on the $F_{r,k}$'s is that they possess exponential moments. This assumption implies the existence of moments of all orders; in practice such an assumption can safely be made in the scheduling context. We let $m_{r,k}$ denote the expected processing time for step (r, k) . We define the bottleneck machine of the job-shop as $\arg \max_{1 \leq i \leq I} \sum_{(r,k) \in C_i} m_{r,k}$. We assume that the bottleneck machine is unique. Our probabilistic assumptions are more general and less restrictive than those employed in most of the papers [14, 11, 34, 30, 19, 9, 22].

Our main result is the following (for $x > 0$, $\lceil x \rceil$ is the smallest integer greater than or equal to x):

Theorem 1.1 Consider random instances of the job-shop with fixed routes. Let N denote its multiplicity. Let T^{Opt} denote the (random) optimal makespan and T^H denote the (random) makespan of the fluid heuristic with safety stocks

$$S_{r,k} = \lceil c_2 \log N \rceil \quad \text{for } k = 2, \dots, K_r, r = 1, \dots, R.$$

There exist constants $c_1 > 0$ and $c_2 > 0$ such that

$$\mathbf{P}\{T^H - T^{Opt} \geq c_1 \log N\} \leq 1/N \quad \text{for all } N \geq 1.$$

Note that Theorem 1.1 implies that if we change c_1 we can make the probability go down faster than $1/N^\kappa$ for any $\kappa \geq 1$:

$$\mathbf{P}\{T^H - T^{Opt} \geq \kappa c_1 \log N\} = \mathbf{P}\{T^H - T^{Opt} \geq c_1 \log N^\kappa\} \leq 1/N^\kappa. \quad (1.2)$$

As a corollary to Theorem 1.1, we have

Corollary 1.2 There exists a constant $c_3 > 0$ such that

$$\mathbf{E}[T^H - T^{Opt}] \leq 2c_1 \log N + c_3 \quad \text{for } N \geq 1.$$

Both Theorem 1.1 and Corollary 1.2 will be proved in Section 3, following the proof of Theorem 3.1.

We note that our fluid heuristic is similar to the *drum buffer rope* scheduling heuristic suggested by Goldratt [16, 17], in which safety stocks (the buffer) are used to keep the bottleneck machine busy and the bottleneck machine (the drum) is used to pace the other machines, and the other machines are not allowed to run ahead (the rope). Goldratt's heuristic is essentially designed to schedule flow-shop production lines, and there is no attempt to evaluate its performance or quantify the buffers, whereas we consider the more general job-shop model and provide a probabilistic analysis.

How good is our heuristic in comparison to other approximation schemes? It has been shown [38] that unless $P = NP$ there does not exist a polynomial time approximation algorithm that constructs a schedule with length guaranteed to be less than $5/4$ the optimal makespan. This excludes the existence of an approximation scheme which is asymptotically optimal as the number of jobs approaches ∞ , uniformly for all problems. Taking a different approach, based on geometrical ideas expressed by the Steinitz Lemma, Sevastyanov [31, 32] (see also [5]) describes a polynomial time algorithm, which achieves additive suboptimality bounded by a quantity proportional to the longest job-step. In our context of a fixed set of routes and multiple jobs on each route, Sevastyanov's algorithm has complexity $(\sum N_r)^2 \bar{K}^2 T^2$, and constructs a schedule with makespan bounded by $T^* + (\bar{K} - 1)(I\bar{K}^2 + 2\bar{K} - 1) \times \max X_{r,k}(j)$, where $\bar{K} = \max K_r$. This is asymptotically optimal if $\max X_{r,k}(j)/T^* \rightarrow 0$ as the problem size increases. In particular, under the assumptions of our probabilistic analysis, $T^* = O(N)$, while $\max X_{r,k}(j) = O(\log N)$, in probability. Hence the bound of Sevastyanov is comparable to the performance of our heuristic. However, our heuristic is much simpler, and can be executed on-line in linear time. Furthermore, there is some computational evidence (see [6, 7]) that the constants multiplying the $O(\log N)$ terms in Sevastyanov's bound are rather large.

This paper is part of an ongoing effort to derive heuristics for solving discrete deterministic scheduling problems and for optimal control of stochastic multiclass queueing networks, based on fluid approximations. The general approach is outlined in [37], where three steps are distinguished: Approximate the discrete (deterministic or stochastic) optimization problem by a continuous deterministic fluid optimization problem. Solve the fluid problem. Translate the fluid solution into a discrete on-line policy. For the problem in this paper, with makespan as the objective, the fluid approximation is simple, it provides a lower bound, and the solution of the fluid problem is straightforward (see Weiss [36]). The main issue in this paper is the translation of the fluid into a discrete policy, and probabilistic analysis to assess the optimality gap which this creates.

Some earlier papers which use similar ideas to the work in this paper are [6, 7] and [4]. These papers deal with the case of job shops with identical jobs on each route, that is $X_{r,k}(j) = X_{r,k}(1), j = 1, \dots, N_r$, and suggest a heuristic similar to ours. Because there is no variability in the processing times one can use safety stocks or offsets which are independent of the multiplicity N , and obtain a suboptimality gap which depends on the basic set of jobs but not on N . Bertsimas and Gamarnik [4] also consider the case of independent N_1, \dots, N_R , and obtain a bound of order $T^H - T^* \leq O(\sqrt{T^*})$. Boudoukh, Penn and Weiss [6, 7] consider also the case of variable jobs on each route, and suggest a more general fluid heuristic for them, which is also applicable to independent N_1, \dots, N_R . We discuss this work briefly in Section 5.

Fluid limit models have recently figured prominently in the theory of multi-class queueing networks [13, 35, 27]. The approximation of job-shops by a fluid model was first suggested by Anderson [2]. Related ideas are discussed in [10, 12, 3, 18, 24, 25].

The rest of the paper is organized as follows: In Section 2 we present the heuristic, which is constructed in three stages. We first reduce the job-shop problem to a re-entrant line scheduling problem by *kitting* together one job from each route. We then present an infeasible *backlog* schedule, which performs jobs on the bottleneck machine with no interruptions, and uses these to initiate *just in time* work at the less tight machines. Next, we introduce a *safety stocks* and *offsets* schedule, in which we first build up safety stocks of jobs ready for processing at each step, and then use an offset backlog schedule which is feasible.

In Section 3 we present the probabilistic analysis of the heuristic. The probability estimates are formulated as general results and proved from first principles, to keep the paper transparent and self contained. In Section 4 we discuss the choice of safety stocks and illustrate it by a simulation. We conclude the paper with Section 5 where a refined fluid heuristic in Boudoukh, Penn and Weiss [6, 7] is discussed.

2 The On-Line Fluid Scheduling Heuristic

We construct the heuristic in three steps. We first reduce the job-shop problem to a re-entrant line scheduling problem which is more restrictive, but possesses the same lower bound. Next we define an infeasible backlog schedule which keeps the bottleneck machine busy. Finally we introduce safety stocks to make the backlog schedule feasible. We borrow the terms *kitting*, *backlog*, *just in time*, *safety stock*, and *offset* from industrial engineering. While our usage may

differ from accepted definitions, the connection to manufacturing and supply chain management should remain obvious. We shall use the following simple example as an illustration throughout the paper.

Example 2.1 (A 3 machine, 2 routes example) A job-shop with $I = 3$ machines and $R = 2$ routes is described schematically in Figure 1. The first route is $2 \rightarrow 1 \rightarrow 2$ (first

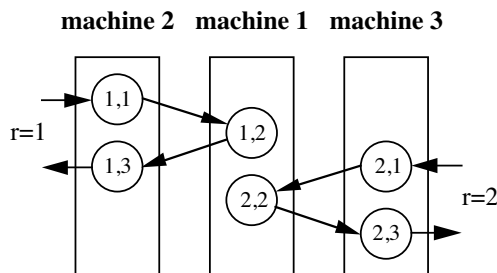


Figure 1: A 3 machine, 2 route, job-shop

step at machine 2, second at machine 1, third at machine 2, that is $\sigma(1,1) = 2, \sigma(1,2) = 1, \sigma(1,3) = 2$). Route 2 is $3 \rightarrow 1 \rightarrow 3$ ($\sigma(2,1) = 3, \sigma(2,2) = 1, \sigma(2,3) = 3$).

Example 2.2 (continued, data for 2×8 jobs) There are $N_1 = N_2 = N = 8$ jobs on each route. The data for the jobs is given in Table 1, which lists the steps, the jobs of each route and the processing times: The processing times were generated randomly from a geometric

Table 1: Processing times for 2×8 jobs

k	Step	$\sigma(k)$	m_k	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	Total
1	(1,1)	2	2	2	2	3	2	1	2	3	2	17
2	(1,2)	1	3	4	3	2	4	3	1	1	8	26
3	(1,3)	2	2	5	3	2	1	5	1	2	2	21
4	(2,1)	3	1	1	1	1	1	1	1	1	1	8
5	(2,2)	1	2	1	2	3	3	2	1	4	2	18
6	(2,3)	3	3	1	6	3	1	3	2	1	1	18

distribution, with expected processing time m_k for the k 'th job-step, $k = 1, \dots, 6$. The machine workloads (as defined in (1.1)) are:

$$T_1 = 44, \quad T_2 = 38, \quad T_3 = 26, \quad T^* = 44.$$

The bottleneck machine is machine 1.

2.1 Reduction to Re-Entrant Line Scheduling Problem

For the purpose of the heuristics we can convert the problem, with R routes and N jobs on each route, to a problem with a single route: We concatenate all the steps of all the routes, in an arbitrary fixed order, say $1, \dots, R$, and then we have a single route job-shop, with steps $k = 1, \dots, K$ where $K = \sum_{r=1}^R K_r$, and with N *kitted* jobs on this route. The lower bound T^* for this newly formed re-entrant line is the same as for the original job-shop, though the actual optimal makespan of the re-entrant line will be greater or equal to that of the job-shop since the problem is more constrained.

The re-entrant line consists of machines $i = 1, \dots, I$, of a single route with ordered steps $k = 1, \dots, K$ where step k is performed by machine $\sigma(k)$ and C_i is the set of steps of machine i , and of jobs $j = 1, \dots, N$ requiring $X_k(j)$ processing times.

Example 2.3 (continued, reduction to a re-entrant line) *The reduction to re-entrant line concatenates routes 1 and 2 into the single, 6 step route, $2 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 3$, as shown in Figure 2.*

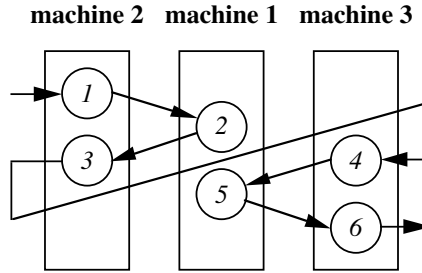


Figure 2: Concatenating the two routes, to a re-entrant line

Re-entrant lines were introduced by Kumar [20, 21], they are quite common in manufacturing systems, and are of interest in their own right; in particular they are used to model the production process in *semi-conductor wafer fabrication*. The problem of scheduling re-entrant lines to minimize makespan is NP-hard (recall that minimizing makespan of a 3 machine flow-shop, which is a re-entrant line with three steps moving through three machines in the order $1 \rightarrow 2 \rightarrow 3$, is in itself an NP-hard problem [23]).

A schedule for the re-entrant line (or for the job-shop) is given by specifying $s_k(j)$ the starting time of step k of job j , for $k = 1, \dots, K$, $j = 1, \dots, N$. The following additional notations and constraints define all feasible schedules:

$$t_k(j) = s_k(j) + X_k(j), \quad (2.1)$$

$$s_k(j) \geq t_{k-1}(j), \quad (2.2)$$

$$I_{k,j}(t) \in \{0, 1\}, \quad (2.3)$$

$$I_{k,j}(t) = 1 \Leftrightarrow s_k(j) \leq t < t_k(j), \quad (2.4)$$

$$I_k(t) = \sum_{j=1}^N I_{k,j}(t), \quad (2.5)$$

$$\sum_{k \in C_i} I_k(t) \leq 1, \quad t \geq 0. \quad (2.6)$$

Here $t_k(j)$ denotes the completion time of step k of job j , and the first constraint (2.1) defines it and specifies that each step is carried out without preemption. $I_{k,j}(t)$ is the indicator that step k of job j is performed at time t , by machine $\sigma(k)$, and it is defined by (2.1)-(2.4). $I_k(t)$ is the indicator that machine $i = \sigma(k)$ is assigned to perform step k (on some job) at time t , as defined by (2.5). The constraint (2.6) ensures that each machine is assigned to no more than one job at a time, and the constraints (2.3)-(2.6) also ensure that machines are assigned for all the processing. The constraint (2.2) requires that the consecutive steps of each job j are carried out one at a time, in the order $1, \dots, K$.

Note that in the reduction from a general job shop to a re-entrant line, we have kitted jobs from different routes to a single job, with $K = \sum_{r=1}^R K_r$ job-steps, and as a result we have reduced the number of jobs from NR to N . The concatenation is enforced by addition of the constraints $s_{r+1,1}(j) \geq t_{r,K_r}(j)$, to the other step constraints of the NR jobs, and their inclusion in (2.2). Hence every feasible schedule of the re-entrant line, satisfying (2.1-2.6) (with the augmented (2.2)), is also a feasible schedule for the original job-shop, with the same makespan.

The makespan of the re-entrant line is bounded by the same machine lower bound in (1.1), which in the kitted re-entrant line notation is:

$$\text{makespan} \geq T^* = \max_{1 \leq i \leq I} \sum_{k \in C_i} \sum_{j=1}^N X_k(j). \quad (2.7)$$

Let $Q_k^+(t)$ denote the number of jobs which have not completed step k by time t , i.e.,

$$Q_k^+(t) = \#\{j : t < t_k(j)\}. \quad (2.8)$$

By the above constraints, in a feasible schedule, $Q_k^+(t)$ is a non-negative integer. It is non-increasing in t , with $Q_k^+(t) \geq Q_{k-1}^+(t)$ for each $t \geq 0$.

2.2 The Backlog Schedule

Once we have reduced the original problem to a re-entrant line in Section 2.1, we describe in this section the backlog schedule. In this schedule we allow a machine to start work on a job step even if the previous step of this job on a different machine has not been completed. Such jobs could be said to be backlogged, hence the name. Note that this is an infeasible schedule, it will be the basis for a feasible schedule in Section 2.3.

A central role in our schedule is played by the bottleneck machine. We consider two models for the definition of the bottleneck machine. Throughout this section (Section 2) we consider a single instance of the problem with processing times $X_k(j)$, and we define the bottleneck machine as one of the machines in the set $\arg \max_i T_i$ (defined in (1.1)). In Section 3 we perform

a probabilistic analysis, and we consider a population of problem instances, each with its own processing times $X_k(j)$, which are drawn randomly from a distribution with mean m_k . Under this second model we define the bottleneck machine as a machine in the set $\arg \max_i \sum_{k \in C_i} m_k$, and we assume that this bottleneck machine is unique. In either case we will for ease of notation label the bottleneck machine as *machine 1*. Note that in the second model, it is possible that for some instances of the problem $T_1 < T^*$. The reader should note that all the results of this section continue to hold even when $T_1 < T^*$.

To describe the backlog schedule let $\{k(i, 1), \dots, k(i, L_i)\} = C_i$, where $k(i, 1) < \dots < k(i, L_i)$ denote the ordered steps of machine i . The backlog schedule is defined as follows: On machine 1 process jobs in the order $1, \dots, N$, perform the steps $k(1, 1), \dots, k(1, L_1)$ in order for each job, and do all this without any interruptions. This will define starting times

$$\begin{aligned} s_{k(1,1)}(1) &= 0, \\ s_{k(1,1)}(j) &= t_{k(1,L_1)}(j-1), \quad j = 2, \dots, N, \\ s_{k(1,l)}(j) &= t_{k(1,l-1)}(j), \quad l = 2, \dots, L_1, \quad j = 1, \dots, N. \end{aligned} \quad (2.9)$$

For the other machines, at the moment that job j is started by machine 1 it is queued up at all the other machines, which work on their queues in FCFS order, and perform all the steps of a job, at each machine, in order and without interruptions, so that for $i = 2, \dots, I$:

$$\begin{aligned} s_{k(i,1)}(1) &= 0, \\ s_{k(i,1)}(j) &= \max\{s_{k(1,1)}(j), t_{k(i,L_i)}(j-1)\}, \quad j = 2, \dots, N, \\ s_{k(i,l)}(j) &= t_{k(i,l-1)}(j), \quad l = 2, \dots, L_i, \quad j = 1, \dots, N. \end{aligned} \quad (2.10)$$

To these definitions of the starting times are added the usual definitions of completion times, for all k, j :

$$t_k(j) = s_k(j) + X_k(j). \quad (2.11)$$

The backlog schedule satisfies all the constraints (2.1)-(2.6) of the re-entrant line scheduling problem in Section 2.1, except for the precedence constraints (2.2) between successive steps which belong to different machines, namely:

$$s_k(j) \geq t_{k-1}(j), \quad \text{where } k \in C_i, k-1 \in C_{i'}, i \neq i', \quad (2.12)$$

may be violated.

Example 2.4 (continued: The backlog schedule) *Machine 1 is taken as the bottleneck machine, and the backlog schedule is constructed by (2.9)-(2.11). Figure 3 shows the Gantt chart (see [29] for a reference on Gantt charts and a photograph of Henry Laurence Gantt) for the backlog schedule.*

Recall definition (2.8) of $Q_k^+(t)$, the number of jobs that have not yet completed step k by the time t . At time t , $N - Q_k^+(t)$ is the number of jobs which have completed step k , and $N - Q_k^+(t) + I_k(t)$ is the number of jobs that have started step k .

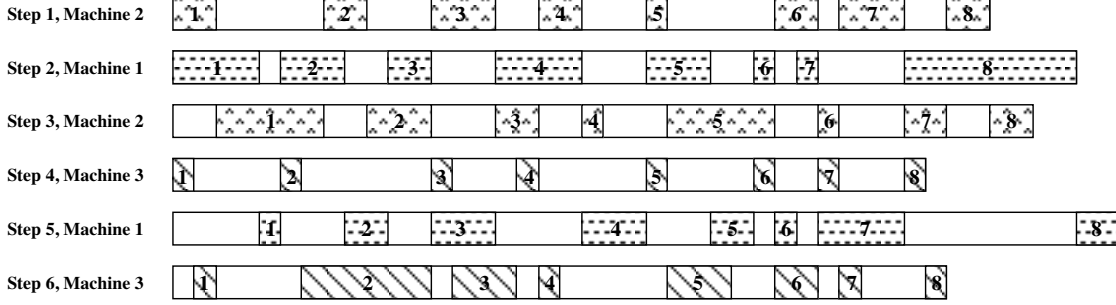


Figure 3: Gantt chart of the backlog schedule

We define the queue at machine i , $i \neq 1$, as:

$$\begin{aligned}
 Z_i(t) &= \text{arrivals to machine } i - \text{departures from machine } i \\
 &= (N - Q_{k(1,1)}^+(t) + I_{k(1,1)}(t)) - (N - Q_{k(i,L_i)}^+(t)) \\
 &= Q_{k(i,L_i)}^+(t) - Q_{k(1,1)}^+(t) + I_{k(1,1)}(t).
 \end{aligned}$$

We define the backlog of step k , $k > 1$, as (here $a^+ = \max(a, 0)$):

$$\begin{aligned}
 B_k(t) &= \left[(N - Q_k^+(t) + I_k(t)) - (N - Q_{k-1}^+(t)) \right]^+ \\
 &= \left(Q_{k-1}^+(t) - Q_k^+(t) + I_k(t) \right)^+.
 \end{aligned}$$

This is the number of jobs for which the backlog schedule has started step k by time t , while step $k - 1$ has not yet been completed.

Example 2.5 (continued: the queues and backlogs) Figures 4,5 show the queues at machines 2,3 and the backlogs of steps $k = 2, \dots, 6$.

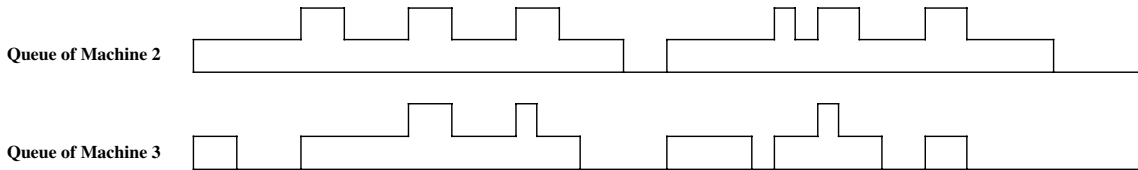


Figure 4: Queue lengths at machines 2 and 3

Lemma 2.1 *The process $\{Z_i(t), 0 \leq t \leq T_1\}$ behaves like the queue length process of a single server queue up to the arrival time of the N th customer, with inter-arrival times $\tau_1 = 0$, $\tau_{j+1} - \tau_j = \sum_{k \in C_1} X_k(j)$, and service times $v_j = \sum_{k \in C_i} X_k(j)$.*

Proof This is immediate from the definition of the backlog schedule, (2.9)-(2.11). \square

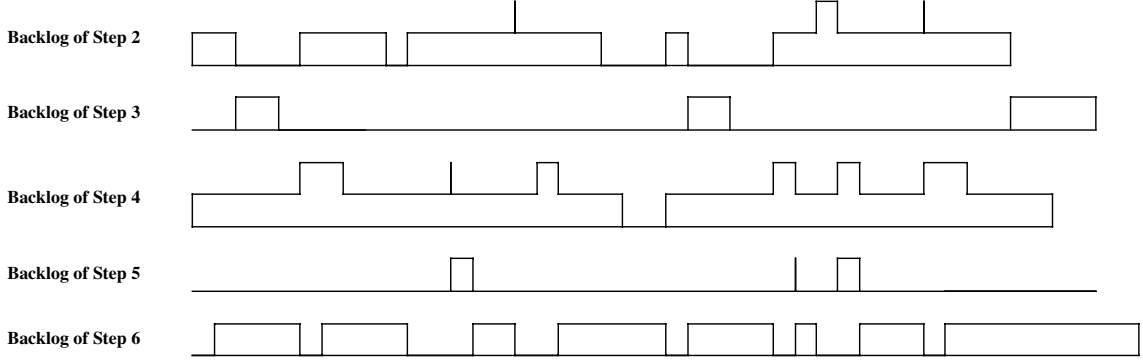


Figure 5: Backlogs of steps $k = 2, \dots, 6$

Lemma 2.2 *For all k and t , the backlog of step k is bounded by the maximum of the queue lengths at all the non-bottleneck machines at the time t , or by 1, that is*

$$B_k(t) \leq \max \left\{ 1, \max_{i \neq 1} \{Z_i(t)\} \right\}.$$

Proof *case (i): $k - 1, k \in C_i \Rightarrow B_k(t) = 0$.*

To see this, let $k = k(i, l), k - 1 = k(i, l - 1)$, and note that: $s_{k(i, l)}(j) = t_{k(i, l - 1)}(j)$, hence at any time t , the number of jobs that started step k equals the number of jobs that completed step $k - 1$, $N - Q_{k(i, l)}^+(t) + I_{k(i, l)}(t) = N - Q_{k(i, l - 1)}^+(t)$, and hence $B_k(t) = 0$.

case (ii): $k - 1 \in C_1, k \in C_i, i \neq 1 \Rightarrow B_k(t) \leq 1$.

To see this, let $k = k(i, l)$, and note that: $s_{k(i, l)}(j) \geq s_{k(i, 1)}(j) \geq s_{k(1, 1)}(j) = t_{k(1, L_1)}(j - 1)$. Hence, at any time t , the number of jobs that have started step k is less or equal to 1 plus the number of jobs that departed machine 1, and therefore: $N - Q_{k(i, l)}^+(t) + I_{k(i, l)}(t) \leq 1 + N - Q_{k(1, 1)}^+(t)$, and hence $B_k(t) \leq 1$.

case (iii): $k - 1 \in C_i, i \neq 1, k \in C_1 \Rightarrow B_k(t) \leq Z_i(t)$.

Let $k = k(1, l'), k - 1 = k(i, l'')$, then at any time t the number of starts of step $k(1, l')$ is less or equal to the number of starts of step $k(1, 1)$, $N - Q_{k(1, l')}^+(t) + I_{k(1, l')}(t) \leq N - Q_{k(1, 1)}^+(t) + I_{k(1, 1)}(t)$. Also at any time t the number of completions of step $k(i, L_i)$ is less or equal to the number of completions of step $k(i, l'')$, $N - Q_{k(i, l'')}^+(t) \geq N - Q_{k(i, L_i)}^+(t)$. Hence,

$$\begin{aligned} B_k(t) &= \left((N - Q_{k(1, l')}^+(t) + I_{k(1, l')}(t)) - (N - Q_{k(i, l'')}^+(t)) \right)^+ \\ &\leq \left((N - Q_{k(1, 1)}^+(t) + I_{k(1, 1)}(t)) - (N - Q_{k(i, L_i)}^+(t)) \right)^+ = Z_i(t). \end{aligned}$$

case (iv): $k - 1 \in C_i, k \in C_{i'}, i \neq i', i, i' \neq 1 \Rightarrow B_k(t) \leq Z_i(t)$.

$$\begin{aligned} B_k(t) &= \left((N - Q_k^+(t) + I_k(t)) - (N - Q_{k-1}^+(t)) \right)^+ \\ &\leq \left((N - Q_{k(1, 1)}^+(t) + I_{k(1, 1)}(t)) - (N - Q_{k-1}^+(t)) \right)^+ \end{aligned}$$

$$+ \left((N - Q_k^+(t) + I_k(t)) - (N - Q_{k(1,1)}^+(t) + I_{k(1,1)}(t)) \right)^+.$$

Repeating the argument of case (ii), it is seen immediately that the second summand is 0. Repeating the argument of case (iii), the first summand is at most $Z_i(t)$. This completes the proof. \square

Lemma 2.3 *The makespan of the backlog schedule satisfies the inequality:*

$$\text{makespan} \leq T^* + \max_{i \neq 1} \left\{ \sum_{j=N-Z_i(T^*)+1}^N \sum_{k \in C_i} X_k(j) \right\}.$$

Proof This is immediate: Machine 1 works for exactly the time T^* . The added time beyond T^* is the time required by the backlog schedule to finish all the steps of all the queued jobs. \square

Lemma 2.4 *Let $\bar{B}_k = \max_t B_k(t)$. Then for all j , and all $k > 1$,*

$$s_k(j + \bar{B}_k) \geq t_{k-1}(j).$$

Proof Assume to the contrary that for some $k > 1$, and some j , $t_{k-1}(j) > s_k(j + \bar{B}_k)$. Consider any t such that $t_{k-1}(j) > t > s_k(j + \bar{B}_k)$. Then

$$N - Q_{k-1}^+(t) \leq j - 1, \quad N - Q_k^+(t) + I_k(t) \geq j + \bar{B}_k,$$

But this implies that $B_k(t) > \bar{B}_k$, which contradicts the definition of \bar{B}_k . \square

2.3 Safety Stocks and Offsets

Safety stocks are used to make the backlog schedule feasible. The safety stocks create offsets, and we apply the backlog schedule to the offset jobs to obtain a feasible schedule. This combination, to be made clear soon, constitutes the fluid heuristic. To examine feasibility we shall examine the set of jobs which have completed step $k - 1$ but have not yet started step k . We define *buffer k* as the set of these jobs, $k = 2, \dots, K$. Jobs which have not yet had any processing at all constitute *buffer 1*.

The *fluid heuristic* consists of four consecutive parts. In the first part we partially process some jobs, to create *safety stocks* in the buffers of the various steps. At the end of this part the first jobs available for processing at each step are *offset*. In the middle part we implement a backlog schedule, on an *offset problem*. This is the *offset backlog schedule* which consists of a period of uninterrupted work at the bottleneck machine (part two), followed by a period of running out queues at the slack machines (part three). Finally we empty the safety stocks (part four).

Note that, similar to the backlog schedule, for each step k the order of processing is $1, \dots, N$.

2.3.1 Creating safety stocks

The first part of the fluid heuristic is to create a safety stock of size S_k at buffer k , for $k = 2, \dots, K$, by partially processing jobs. The remaining $\tilde{N} = N - \sum_{k=2}^K S_k$ unstarted jobs are in buffer 1, waiting for step 1.

To create the safety stocks we process steps $1, \dots, K-1$ of jobs $1 \leq j \leq S_K$, steps $1, \dots, K-2$ of jobs $S_K + 1 \leq j \leq S_K + S_{K-1}$, etc, steps $1, \dots, k-1$ of jobs $\sum_{k'=k+1}^K S_{k'} + 1 \leq j \leq \sum_{k'=k}^K S_{k'}$, ending with step 1 on jobs $\sum_{k'=3}^K S_{k'} + 1 \leq j \leq \sum_{k'=2}^K S_{k'}$.

Let $j_k = \sum_{k'=k+1}^K S_{k'} + 1$ denote the index of the first (lowest index) job contained in the safety stock at buffer k , $k = 1, \dots, K$. This is the offset of jobs in buffer k .

The creation of the safety stocks will take a time T_0 , where

$$T_0 \leq \sum_{k=2}^K \sum_{j=j_k}^{j_k+S_k-1} \sum_{k'=1}^{k-1} X_{k'}(j). \quad (2.13)$$

In particular, this upper bound on T_0 will be achieved by a schedule which keeps exactly 1 machine busy throughout the time that the safety stocks are created.

2.3.2 An offset backlog schedule

We next define a new problem, called the *offset problem*, with the original route, with $\tilde{N} = N - \sum_{k=2}^K S_k$ jobs, and with processing times given by:

$$\tilde{X}_k(j) = X_k(j + j_k - 1) \quad 1 \leq j \leq \tilde{N}. \quad (2.14)$$

For the offset problem we create the backlog schedule, and obtain starting and completion times, denoted by $\tilde{s}_k(j), \tilde{t}_k(j)$. Let $\tilde{Z}_i(t)$ denote the queue at machine i and $\tilde{B}_k(t)$ denote the backlog at step (buffer) k , for this backlog schedule.

The total duration of the backlog schedule of the offset problem is, by Lemma 2.3, $\tilde{T}^* + \tilde{T}_1$ where \tilde{T}^* is processing time of the bottleneck machine 1, and \tilde{T}_1 is the additional time needed to run out the queues at the other machines, at time \tilde{T}^* .

Return to the original problem, and define:

$$s_k(j) = T_0 + \tilde{s}_k(j - j_k + 1), \quad j_k \leq j \leq \tilde{N} + j_k - 1. \quad (2.15)$$

This defines starting and completion times for \tilde{N} jobs at each of the steps. To be precise, for step k it defines starting and completion times of this step for the jobs $j_k, \dots, j_k + \tilde{N} - 1$. All these starting times are larger than T_0 .

Note that while the backlog schedule for the offset problem schedules a complete *cycle* consisting of all the steps of job j to run more or less simultaneously on all the machines, when we re-translate these times to the original problem, we are actually scheduling steps of different jobs in each cycle, namely jobs $j + j_k - 1, k = 1, \dots, K$ at cycle j . We call this an *offset* schedule, since we created an offset between the jobs that are scheduled at step $k-1$ and step k . In fact buffers $k-1$ and k are offset by $j_{k-1} - j_k = S_k$.

Example 2.6 (continued: The offset backlog schedule) Assume that we have built up a safety stock of 2 jobs at each of the buffers 2, ..., 6. The initial jobs at the various buffers are $j_1 = 11, j_2 = 9, j_3 = 7, j_4 = 5, j_5 = 3, j_6 = 1$. We assume that from time T_0 onwards the processing times are the same as in Table 1. Hence the processing on the various machines will be the same as in the backlog schedule, described in Figure 3 but now the actual jobs to which the various job-steps belong are different, in fact instead of j we have $j_k + j$. Figure 6 shows the Gantt chart for the offset backlog schedule, scheduling 8 job cycles. Note that together with the safety stocks, whose buildup and depletions are not shown, this examples involves a total of 18 jobs. Note also that none of the processing times of steps of the same job overlap, so this schedule is feasible, and it keeps machine 1 busy for the entire 8 cycles.

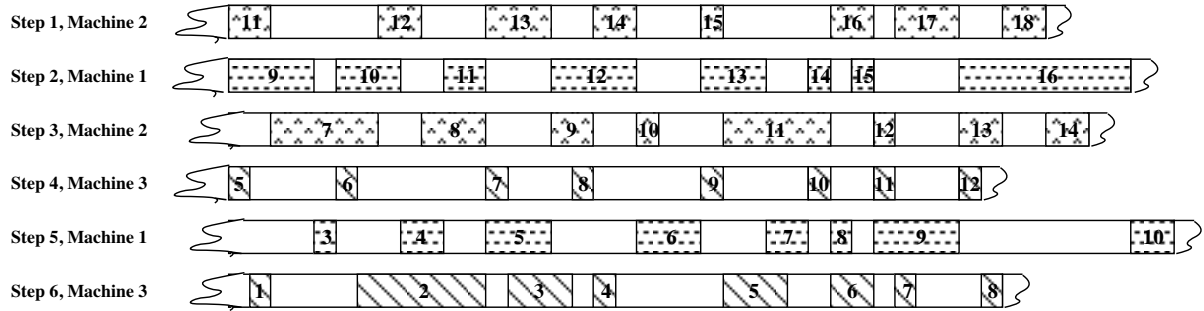


Figure 6: Gantt chart of the offset backlog schedule, using safety stocks

Lemma 2.5 *If for all $k > 1$ and all $0 < t < \tilde{T}^* + \tilde{T}_1$, the backlog schedule for the offset problem satisfies that $S_k \geq \tilde{B}_k(t)$, then the schedule obtained by the fluid heuristic, up to time $T_0 + \tilde{T}^* + \tilde{T}_1$, is feasible.*

Proof We need to show that we always complete previous steps before starting next steps (the constraint (2.2)). Consider some $k > 1$. We need to show that for $j \leq \tilde{N} + j_k - 1$, $s_k(j) \geq t_{k-1}(j)$. The constraints hold for $j < j_k$, because these job starts happen prior to T_0 , and we use a feasible schedule to create the safety stocks. For $j_k \leq j < j_{k-1}$ we have $t_{k-1}(j) \leq T_0$ while $s_k(j) \geq T_0$. For $j \geq j_{k-1}$, the inequality $s_k(j) \geq t_{k-1}(j)$ is equivalent to $\tilde{s}_k(j - j_k + 1) \geq \tilde{t}_{k-1}(j - j_{k-1} + 1)$, which is equivalent to $\tilde{s}_k(j' + S_k) \geq \tilde{t}_{k-1}(j')$ for $j' = j - j_{k-1} + 1$. We now use the assumption $S_k \geq \tilde{B}_k(t)$ and apply Lemma 2.4 to the offset problem, to obtain that indeed $\tilde{s}_k(j' + S_k) \geq \tilde{t}_{k-1}(j')$. This completes the proof. \square

2.3.3 Emptying the buffers

Lemma 2.6 *At the end of the offset backlog schedule, at time $T_0 + \tilde{T}^* + \tilde{T}_1$, buffer k contains S_k jobs, $k = 2, \dots, K$, while buffer 1 is empty.*

Proof At time T_0 buffer 1 contains \tilde{N} jobs, and the other buffers contain S_k jobs. During the period of the offset backlog schedule, step k is performed on \tilde{N} jobs, hence buffer 1 loses

\tilde{N} jobs and is empty at the end, while all the other buffers lose \tilde{N} to the next buffer and gain \tilde{N} jobs from the previous buffer, and end up with S_k jobs. \square

It follows from the lemma that the last part of the fluid heuristic consists of completing the remaining steps of the S_k jobs in buffers $2, \dots, K$. We denote its duration by T_2 , and it is bounded by:

$$T_2 \leq \sum_{k=2}^K \sum_{j=j_k+\tilde{N}}^{j_k+\tilde{N}+S_k-1} \sum_{k'=k}^K X_{k'}(j). \quad (2.16)$$

2.3.4 A final modification of the fluid heuristic

For any instance of the scheduling problem we can choose the safety stocks large enough to make the schedule feasible. However, since we want an on-line schedule, we would like to fix S_k in advance, without looking at all the processing time data. As a result, the heuristic schedule described so far may become infeasible.

To obtain a feasible schedule in all cases, we modify the heuristic as follows: If at any time the start of a step for a particular job is infeasible, because the previous step of this job has not been completed yet, then from that time onwards we use a feasible schedule which engages exactly one machine at all times, until all work is complete. We denote the length of the schedule in that case by T_{long} . We refer to the complete heuristic, with this modification, as the *fluid heuristic*.

2.4 On-Line Implementation of the Heuristic

The heuristic works as follows:

Concatenate all the job-steps of all the routes into a consistent *kitted* job cycle with ordered steps $k = 1, \dots, K$.

Locate the bottleneck machine (use known values of m_k , or calculate $\sum_j X_k(j)$ or use a sample of the j 's). Without loss of generality assume this is machine 1.

Use processing time information and the value of N to determine the size of the safety stocks, S_k .

Create the safety stocks, by some feasible schedule, which keeps at least one machine busy at all times (duration T_0).

Perform \tilde{N} *offset processing cycles* as follows: For cycle j on the bottleneck machine start steps $k \in C_1$ in order, and for each step k , perform that step on job $j + j_k - 1$, without any interruptions. On machine $i = 2, \dots, I$ queue up the work of steps $k \in C_i$ in order, with step k to be performed on job $j + j_k - 1$, and process the queue at machine i in FIFO order. The total processing time on machine 1, for these \tilde{N} cycles is \tilde{T}^* .

After the completion of all \tilde{N} cycles on the bottleneck machine, run out the queues at the other machines, with no interruptions (duration \tilde{T}_1).

Complete the processing of all the remaining safety stocks at all the buffers (duration T_2).

If the schedule becomes infeasible at any time revert to a feasible schedule that keeps at least one machine busy at all times (duration T_{long})

The following proposition is not used in subsequent derivations. It derives a simple bound on the buffer contents throughout the schedule.

Proposition 2.7 *The contents of buffer k never exceed $S_k + \max_t \tilde{Z}_i(t)$, where $i = \sigma(k)$ and $k = 2, \dots, K$.*

Proof The contents of buffer k at time $T_0 + t$ are $S_k + (\tilde{N} - \tilde{Q}_{k-1}^+(t)) - (\tilde{N} - \tilde{Q}_k^+(t))$. Using a similar estimate to that in the proof of Lemma 2.2, case (iv), we have

$$\begin{aligned} (\tilde{N} - \tilde{Q}_{k-1}^+(t)) - (\tilde{N} - \tilde{Q}_k^+(t)) &= \left((\tilde{N} - \tilde{Q}_{k(1,1)}^+(t) + \tilde{I}_{k(1,1)}(t)) - (\tilde{N} - \tilde{Q}_k^+(t)) \right) \\ &+ \left((\tilde{N} - \tilde{Q}_{k-1}^+(t)) - (\tilde{N} - \tilde{Q}_{k(1,1)}^+(t) + \tilde{I}_{k(1,1)}(t)) \right) \leq \tilde{Z}_i(t), \end{aligned}$$

which proves the proposition. \square

3 Probabilistic Analysis of the Heuristic

In this section we perform a probabilistic analysis of the fluid heuristic described in Section 2. For the probabilistic analysis we assume that the processing times $X_k(j)$ are randomly and independently drawn from some distributions. We assume that $\{X_k(j), j = 1, \dots, N\}$ is an iid sequence of positive random variables with distribution F_k for $k = 1, \dots, K$, and different sequences are independent. We need two further assumptions for the analysis:

Our first assumption is that the processing time distributions possess exponential moments, i.e., there exists $\theta > 0$ such that

$$\mathbf{E} \left[e^{\theta \sum_{k=1}^K X_k(1)} \right] < \infty. \quad (3.1)$$

This implies existence of moments of all orders. Denote $m_k = \mathbf{E}[X_k(1)]$, $k = 1, \dots, K$.

The other assumption is that machine 1 is the unique bottleneck machine on the average:

$$\sum_{k \in C_1} m_k > \sum_{k \in C_i} m_k, \quad i = 2, \dots, I \quad (3.2)$$

An instance of the problem therefore has some values realized by the random variables $X_k(j)$. These values become known to the scheduler, and they define the following quantities, which are therefore also random variables:

T^{Opt} is the actual minimum value of the makespan. As a minimum of a finite number of non-idling feasible schedules it is well defined, though it will in practice almost always remain unknown.

T^* is the lower bound defined in (1.1,2.7). Recall that this equals the maximum over the machines of the total workload of each machine. Often this maximum will be obtained at machine 1, though with some small probability it will be obtained on another machine.

T^H is the makespan of the on-line fluid heuristic schedule for the problem. In the implementation of the on-line heuristic we take machine 1 as the bottleneck machine, regardless of whether it is actually, for the particular realization of $X_k(j)$, the most loaded machine.

We define the random quantity:

$$T_\epsilon^N = T^H - T^* \geq T^H - T^{Opt}. \quad (3.3)$$

T_ϵ^N is an upper bound on the sub-optimality of the heuristic schedule. Clearly, $T_\epsilon^N \geq 0$.

The main result of this section is Theorem 3.1. Two lemmas needed for its proof are presented at the end of this section.

Theorem 3.1 *There are constants $c_1 > 0$ and $c_2 > 0$ such that*

$$\mathbf{P}(T_\epsilon^N \geq c_1 \log N) \leq 1/N \quad \text{for all } N \geq 1, \quad (3.4)$$

where T_ϵ^N is the upper bound on the sub-optimality defined in (3.3) for the on-line fluid heuristic with safty stocks

$$S_k = \lceil c_2 \log N \rceil \quad \text{for } k = 2, \dots, K.$$

Proof When $N = 1$, (3.4) holds for any positive constants c_1 and c_2 . Thus, it is enough to prove Theorem 3.1 for $N \geq 2$. For the remainder of this proof, we assume that $N \geq 2$.

Recall the definitions of \tilde{T}^* , \tilde{T}_1 and $\tilde{B}_k(t)$ defined in Section 2.3.2. Define the event:

$$A = \{S_k \geq \tilde{Z}_i(t) \quad \text{for } i = 2, \dots, I, k = 2, \dots, K \text{ and } t \geq 0\}.$$

By Lemma 2.2,

$$\tilde{B}_k(t) \leq \max_{i \neq 1} \tilde{Z}_i(t) \quad \text{for all } t \geq 0 \quad \text{and } k = 1, \dots, K.$$

Hence

$$A \subseteq \{S_k \geq \tilde{B}_k(t) \quad \text{for } k = 2, \dots, K \text{ and } t \geq 0\}.$$

Therefore, by Lemma 2.5, the offset backlog schedule, which forms the central part of the schedule constructed by the fluid heuristic, is feasible on the set A . Denote by A^c the complement of A , and by $1_A, 1_{A^c}$ the indicators of these events.

Using the notation of the Section 2.4 we have

$$T^H \leq (T_0 + \tilde{T}^* + \tilde{T}_1 + T_2) \times 1_A + T_{long} \times 1_{A^c}.$$

Note that $\tilde{T}^* \leq \sum_{k \in C_1} \sum_{j=1}^N X_k(j) \leq T^*$. Hence,

$$T_\epsilon^N \leq (T_0 + \tilde{T}_1 + T_2) \times 1_A + T_{long} \times 1_{A^c}, \quad T_\epsilon^N \times 1_A \leq T_0 + \tilde{T}_1 1_A + T_2.$$

Observe that

$$\begin{aligned}
\mathbf{P}(T_\epsilon^N \geq c_1 \log N) &= \mathbf{P}(\{T_\epsilon^N \geq c_1 \log N\} \cap A) + \mathbf{P}(\{T_\epsilon^N \geq c_1 \log N\} \cap A^c) \\
&\leq \mathbf{P}(T_\epsilon^N 1_A \geq c_1 \log N) + \mathbf{P}(A^c) \\
&\leq \mathbf{P}(T_0 + \tilde{T}_1 1_A + T_2 \geq c_1 \log N) + \mathbf{P}(A^c) \\
&\leq \mathbf{P}(T_0 + T_2 \geq c_3 \log N) + \mathbf{P}(\tilde{T}_1 1_A \geq c_4 \log N) \\
&\quad + \mathbf{P}(A^c)
\end{aligned}$$

for any positive constants c_3 and c_4 with $c_3 + c_4 = c_1$.

For the remainder of the proof, we use Lemmas 3.2 and 3.3 to show that there exist constants c_2 , c_3 and c_4 such that for all $N \geq 2$

$$\begin{aligned}
\mathbf{P}(A^c) &< 1/(3N) && \text{Lemma 3.2} \\
\mathbf{P}(T_0 + T_2 \geq c_3 \log N) &< 1/(3N) && \text{Lemma 3.3} \\
\mathbf{P}(\tilde{T}_1 1_A \geq c_4 \log N) &< 1/(3N) && \text{Lemma 3.3}
\end{aligned}$$

To invoke these lemmas, we examine each of the expressions in turn.

We look first at $\mathbf{P}(A^c)$. We have

$$A^c = \bigcup_{i \neq 1} \left\{ \max_{0 \leq t \leq \tilde{T}^*} \tilde{Z}_i(t) > \lceil c_2 \log N \rceil \right\}.$$

By Lemma 2.1, for $i = 2, \dots, I$, $\tilde{Z}_i(t)$ behaves like the queue length in a single server queue with inter-arrival times $\tau_1 = 0$, $u_j = \tau_{j+1} - \tau_j = \sum_{k \in C_1} X_k(j + j_k)$, and service times $v_j = \sum_{k \in C_i} X_k(j + j_k)$. Notice that $\tilde{T}^* = \tau_{\tilde{N}+1}$. By our assumption (3.1) these inter-arrival and service times possess exponential moments, and by the assumption (3.2) $\mathbf{E}[u_j] > \mathbf{E}[v_j]$. Hence all the conditions of Lemma 3.2 hold, and we can find an appropriate c_2 to make

$$\mathbf{P}\left(\max_{0 \leq t \leq \tilde{T}^*} \tilde{Z}_i(t) > \lceil c_2 \log N \rceil\right) \leq 1/(3(I-1)N).$$

Thus, we have $\mathbf{P}(A^c) \leq 1/(3N)$.

We look next at $\mathbf{P}(T_0 + T_2 \geq c_3 \log N)$. Recall from Lemma 2.6, and equations (2.13) and (2.16), that

$$\begin{aligned}
T_0 + T_2 &\leq \sum_{k=2}^K \sum_{j=j_k}^{j_k+S_k-1} \sum_{k'=1}^{k-1} X_{k'}(j) + \sum_{k=2}^K \sum_{j=j_k+\tilde{N}}^{j_k+\tilde{N}+S_k-1} \sum_{k'=k}^K X_{k'}(j) \\
&\stackrel{D}{=} \sum_{k=2}^K \sum_{j=j_k}^{j_k+S_k-1} \sum_{k'=1}^K X_{k'}(j) \stackrel{D}{=} \sum_{j=1}^{\bar{S}} \sum_{k=1}^K X_k(j)
\end{aligned}$$

where $\stackrel{D}{=}$ denotes equal in distribution and $\bar{S} = \sum_{k=2}^K S_k = (K-1)\lceil c_2 \log N \rceil$. Since $\sum_{k=1}^K X_k(j)$ has finite exponential moments, by Lemma 3.3,

$$\mathbf{P}(T_0 + T_2 \geq c_3 \log N) < 1/(3N),$$

where $c_3 = c c'$, and c and c' are the two constants in Lemma 3.3.

Finally we look at $\mathbf{P}(\tilde{T}_1 1_A \geq c_4 \log N)$. By Lemma 2.3, and by equation (2.14), the processing of all the queued jobs starts at time $T_0 + \tilde{T}^*$, and lasts a time \tilde{T}_1 satisfying

$$\tilde{T}_1 \leq \max_{i \neq 1} \left\{ \sum_{k \in C_i} \sum_{j=\tilde{N}-Z_i(\tilde{T}^*)+1}^{\tilde{N}} \tilde{X}_k(j) \right\} = \max_{i \neq 1} \left\{ \sum_{k \in C_i} \sum_{j=\tilde{N}-Z_i(\tilde{T}^*)+1}^{\tilde{N}} X_k(j + j_k - 1) \right\}.$$

Furthermore, on A we have $Z_i(\tilde{T}^*) \leq \lceil c_2 \log N \rceil$, and therefore:

$$\begin{aligned} \tilde{T}_1 1_A &\leq \max_{i \neq 1} \left\{ \sum_{k \in C_i} \sum_{j=\tilde{N}-\lceil c_2 \log N \rceil+1}^{\tilde{N}} X_k(j + j_k - 1) \right\} \\ &\leq \sum_{i=1}^I \left\{ \sum_{k \in C_i} \sum_{j=\tilde{N}-\lceil c_2 \log N \rceil+1}^{\tilde{N}} X_k(j + j_k - 1) \right\} \\ &\stackrel{D}{=} \sum_{j'=1}^{\lceil c_2 \log N \rceil} \sum_{k=1}^K X_k(j') \end{aligned}$$

Hence

$$\mathbf{P}(\tilde{T}_1 1_A \geq c_4 \log N) \leq \mathbf{P}\left(\sum_{j'=1}^{\lceil c_2 \log N \rceil} \sum_{k=1}^K X_k(j') > c_4 \log N \right)$$

which, again by Lemma 3.3, can be made less than $1/(3N)$. This completes the proof. \square

Theorem 1.1 for general fixed routes follows immediately from Theorem 3.1 by first reducing the case to $N_1 = \dots = N_R = N$, and then reducing the case further to a re-entrant line.

Proof of Corollary 1.2. We first note that, from the construction of the heuristic, there exists a native upper bound on its makespan. It is obtained by allowing only one machine to work at a time. By using the independence of the processing times, there is a constant c_3 such that

$$\left(\mathbf{E}[T^H]^2 \right)^{1/2} \leq c_3 N \quad \text{for } N \geq 1.$$

Now, we have

$$\begin{aligned} \mathbf{E}[T^H - T^{Opt}] &= \mathbf{E} \left[\left(T^H - T^{Opt} \right) 1_{\{T^H - T^{Opt} \leq 2c_1 \log N\}} \right] \\ &\quad + \mathbf{E} \left[\left(T^H - T^{Opt} \right) 1_{\{T^H - T^{Opt} > 2c_1 \log N\}} \right] \\ &\leq 2c_1 \log N + \mathbf{E} \left[\left(T^H - T^{Opt} \right) 1_{\{T^H - T^{Opt} > 2c_1 \log N\}} \right] \end{aligned}$$

$$\begin{aligned}
&\leq 2c_1 \log N + \mathbf{E} \left[T^H \mathbf{1}_{\{T^H - T^{Opt} > 2c_1 \log N\}} \right] \\
&\leq 2c_1 \log N + \left(\mathbf{E} \left[T^H \right]^2 \mathbf{E} \left[\mathbf{1}_{\{T^H - T^{Opt} > 2c_1 \log N\}} \right]^2 \right)^{1/2} \\
&= 2c_1 \log N + \left(\mathbf{E} \left[T^H \right]^2 \left[P\{T^H - T^{Opt} > 2c_1 \log N\} \right] \right)^{1/2} \\
&\leq 2c_1 \log N + \left(\mathbf{E} \left[T^H \right]^2 \left[1/N^2 \right] \right)^{1/2} \\
&= 2c_1 \log N + c_3,
\end{aligned}$$

thus proving the corollary. \square

We now present the two lemmas which justify the three bounds in the proof of Theorem 3.1. These are standard results in probability theory. We present the complete proofs here for three reasons: (1) The results not in standard form and are not easily found in the literature. (2) Our proofs are from first principles, and should emphasize the simplicity of our results. (3) These proofs make the paper self contained.

The first lemma obtains a probabilistic bound on the maximal queue length when a single server queue serves a total of n customers.

Lemma 3.2 *Consider a GI/GI/1 queue. Let $\{u_i, i \geq 1\}$ be iid inter-arrival times and $\{v_i, i \geq 1\}$ be iid service times. Assume that for some $\theta > 0$,*

$$\mathbf{E}[e^{\theta(u_1+v_1)}] < \infty \tag{3.5}$$

and $\mathbf{E}[u_1] > \mathbf{E}[v_1]$. Let

$$\tau_n = u_1 + \dots + u_n$$

be the arrival time of the n th job. Let $Z(t)$ be the queue length (including possibly the one being serviced) at time t . For any $\kappa \geq 1$, there exists a constant $c > 0$ such that for all $n \geq 2$,

$$\mathbf{P} \left(\sup_{0 \leq t \leq \tau_n} Z(t) > c \log n \right) \leq 1/(\kappa n).$$

In particular, for all $n \geq 2$,

$$\mathbf{P} \left(\sup_{0 \leq t \leq \tau_i} Z(t) > c \log n \right) \leq 1/(\kappa n) \quad \text{for any } 1 \leq i \leq n.$$

Proof For each $\theta > 0$, set $f(\theta) \equiv \mathbf{E}[e^{\theta(v_1-u_1)}]$. Since $\mathbf{E}[u_1] > \mathbf{E}[v_1]$ and (3.5) holds, there exists $\theta = \theta_0 > 0$ such that $f(\theta_0) < 1$. (See, for example, Shwartz and Weiss [33, Exercise 1.3].)

We first claim that for any $m \geq 0$ and $l \geq 0$, and $k \geq 1$,

$$\mathbf{P}(u_m + \dots + u_{m+l} + \dots + u_{m+l+k} < v_m + \dots + v_{m+l}) \leq (f(\theta_0))^l \mathbf{E}[e^{-\theta_0 u_1}]^k.$$

To see this, for any $\theta > 0$,

$$\begin{aligned}
& \mathbf{P}(u_m + \dots + u_{m+l} + \dots + u_{m+l+k} < v_m + \dots + v_{m+l}) \\
&= \mathbf{P}\left(v_m - u_m + \dots + v_{m+l} - u_{m+l} - u_{m+l+1} - \dots - u_{m+l+k} > 0\right) \\
&= \mathbf{P}\left(\exp(\theta(v_m - u_m + \dots + v_{m+l} - u_{m+l} - u_{m+l+1} - \dots - u_{m+l+k})) > 1\right) \\
&\leq \mathbf{E}\left[\exp(\theta(v_m - u_m + \dots + v_{m+l} - u_{m+l} - u_{m+l+1} - \dots - u_{m+l+k}))\right] \\
&= \left(\mathbf{E}[e^{\theta v_1}] \mathbf{E}[e^{-\theta u_1}]\right)^l \mathbf{E}[e^{-\theta u_1}]^k,
\end{aligned}$$

where the inequality follows from Chebyshev's inequality (see, for example, Theorem A.113 in Shwartz and Weiss [33]). Next, we have for any $k \geq 1$,

$$\mathbf{P}\left(\bigcup_{l,m=1}^n \{u_m + \dots + u_{m+l} + \dots + u_{m+l+k} < v_m + \dots + v_{m+l}\}\right) \leq n^2 \mathbf{E}[e^{-\theta_0 u_1}]^k.$$

To prove the lemma, we notice that in order for queue length to exceed some constant $c(n)$, it must do so in some busy period. Suppose that the busy period starts at the arrival of the m th job and that the queue length exceeds $c(n)$ for the first time when the $(\ell + m)$ th job is in service. Then the arrival time of the $(m + \ell + c(n))$ th job happens before the service completion of the $(m + \ell)$ th job. Setting

$$c(n) = \lceil -3 \log(\kappa n) / \log \mathbf{E}[e^{-\theta_0 u_1}] \rceil,$$

one can check that

$$\begin{aligned}
& \mathbf{P}\left(\sup_{0 \leq t \leq \tau_n} Z(t) > c(n)\right) \\
&\leq \mathbf{P}\left(\bigcup_{l,m=1}^n \{u_m + \dots + u_{m+l} + \dots + u_{m+l+c(n)} < v_m + \dots + v_{m+l}\}\right) \\
&\leq n^2 \mathbf{E}[e^{-\theta_0 u_1}]^{c(n)} \\
&\leq 1/(\kappa n).
\end{aligned}$$

The lemma is proved by choosing c such that $c \log n \geq c(n)$ for all $n \geq 2$. \square

The second lemma bounds in probability the sum of $\log n$ iid random variables, and is no more than a form of the weak law of large numbers.

Lemma 3.3 *Let $\{Y_i, i \geq 1\}$ be a sequence of iid random variables with $\mathbf{E}[e^{\theta|Y_1|}] < \infty$ for some $\theta > 0$. For any $c > 0$, there exists a constant $c' > 0$ such that*

$$\mathbf{P}\left(\sum_{j=1}^{\lceil c \log n \rceil} Y_j \geq c' \lceil c \log n \rceil\right) \leq 1/(3n) \quad \text{for } n \geq 1. \tag{3.6}$$

Proof

$$\begin{aligned} \mathbf{P}\left(\sum_{j=1}^{\lceil c \log n \rceil} Y_j \geq c' \lceil c \log n \rceil\right) &\leq e^{-c' \lceil c \log n \rceil \theta} \mathbf{E}\left[e^{\theta(\sum_{j=1}^{\lceil c \log n \rceil} Y_j)}\right] \\ &= \left(e^{-c' \theta} \mathbf{E}[e^{\theta Y_1}]\right)^{\lceil c \log n \rceil}. \end{aligned}$$

For c' large enough, $e^{-c' \theta} \mathbf{E}[e^{\theta Y_1}] \leq 1$, hence,

$$\left(e^{-c' \theta} \mathbf{E}[e^{\theta Y_1}]\right)^{\lceil c \log n \rceil} \leq \left(e^{-c' \theta} \mathbf{E}[e^{\theta Y_1}]\right)^{c \log n}.$$

Hence we choose c' such that

$$\left(e^{-c' \theta} \mathbf{E}[e^{\theta Y_1}]\right)^{c \log n} \leq 1/(3n) \quad \text{for all } n \geq 2.$$

□

4 Implementation of the Fluid Heuristic: Size of Safety Stocks

The probabilistic analysis demonstrates that a fluid heuristic gets asymptotically close to the lower machine workload bound with high probabilities. To achieve the asymptotics we could be quite profligate, and for the sake of simplicity impose concatenation of all the routes, and use a uniform safety stock level for all the buffers. In implementing the heuristic we can do somewhat better, and we discuss this now.

First, retaining the re-entrant line structure, we can use individual safety stock sizes S_k , where by the proof of Lemma 2.2 what we require is:

If $k - 1, k \in C_i$ we can take $S_k = 0$.

If $k - 1 \in C_1, k \in C_i, i > 1$ we can take $S_k = 1$.

If $k - 1 \in C_i, k \in C_{i'}, i' \neq i$ we need to take S_k such that $S_k > Z_i(t)$, i.e., the safety stock exceeds the queue length at machine i .

Safety stocks are most important for the buffers of the bottleneck machine.

By Lemmas 2.2 and 2.5 if we wish to utilize the bottleneck machine fully during the \tilde{N} cycles of the offset backlog schedule, we should have $S_k > Z_i(t)$. To achieve this the safety stocks S_k should be as large as the maximum queue length during the processing of the first \tilde{N} customers of a GI/G/1 queue with traffic intensity $\rho = T_i/T^*$. In fact, for a prior probability bound as guaranteed by Theorem 3.1, the constants c_1, c_2 can be calculated, but they may not be very useful in setting the size of the safety stocks, since the bounds in the Lemmas 3.2 and 3.3 are not sharp. In practice S_k should be determined by direct study, e.g. by simulation, of the queues with inter-arrival times $\sum_{k \in C_1} X_k(j)$ and service times $\sum_{k \in C_i} X_k(j)$ for $j = 1, \dots, N$, where $i = 2, \dots, I$.

Finally we remark on the difference between job-shop with several routes, and the re-entrant line to which we reduced it in Section 2.1. Consider the case of R routes with $N_1 = \dots = N_R = N$. We do not need to create safety stocks at the buffers of job-steps $(r, 1)$, and the construction of the safety stocks at the other buffers requires much less processing then, since the safety stocks at buffer (r, k) require processing of job-steps $(r, 1), \dots, (r, k - 1)$ only.

Example 4.1 (continued: The offset backlog schedule with smaller safety stocks) *Because steps 1, 2, 3 belong to the first route, 4, 5, 6 to the second, we need to process steps 1, 2 for S_3 jobs and step 1 for S_2 jobs to create the required safety stocks on the first route, and we need to process steps 4, 5 for S_6 jobs and step 4 for S_5 jobs to create the required safety stocks on the second route. We need $S_3 = S_6 = 1$ since these steps follow the bottleneck machine. Examination of the queues in Figure 4 shows that we need no more than 2 jobs in any buffer. In fact the buffer of step 5 needs no more than one job since its backlog (see Figure 5) never exceeds 1. Hence we can start the offset backlog schedule at time T_0 with offsets $j_1 = 4, j_2 = 2, j_3 = 1$ on the first route, and offsets $j_4 = 3, j_5 = 2, j_6 = 1$. The offset backlog schedule of the 8 cycles in the middle part of the schedule are given in Figure 7, and it is easily seen that this is a feasible schedule.*

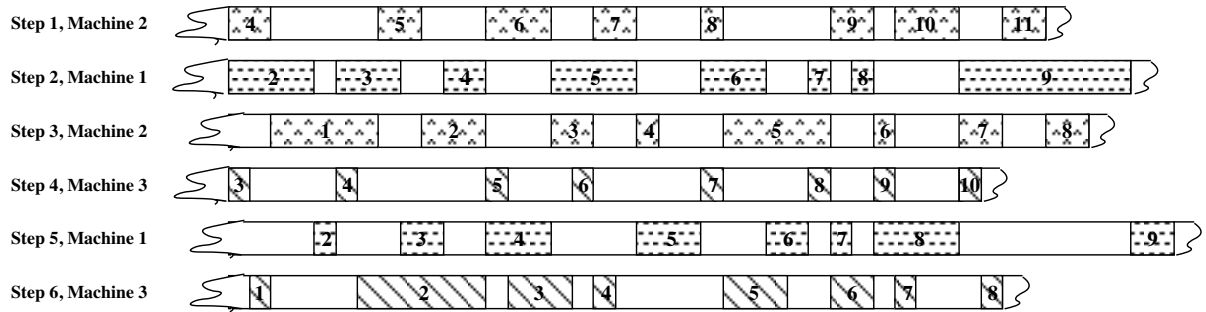


Figure 7: Gantt chart of the offset backlog schedule, with modified safety stocks

Example 4.2 (continued and concluded: Required safety stocks) *We simulated the job-shop of the example, running the backlog schedule, for varying values of N . Recall that the processing times for this example were generated from geometric distributions, with means $m_k, k = 1, \dots, 6$ given in Table 1. We obtained the maximal queue lengths $Z_2(t), Z_3(t)$ at machines 2, 3 and the backlogs $B_k(t)$ at buffers 2 and 5. These indicate the required safety stocks which would make the offset backlog schedule feasible for the various values of N . The results of a sample of 100 replications are given in Table 2. The symbol \bar{Z}_2 denotes the maximum queue size at station 2, namely, $\bar{Z}_2 = \max_{t \geq 0} Z_2(t)$. It is a random variable. There are three rows in the table that correspond to \bar{Z}_2 . The first row are sample values of \bar{Z}_2 for the various N obtained from one particular realization of processing times (replication 50 in our simulation runs). The second row are the sample averages of \bar{Z}_2 , averaged over the 100 replications. The third row are the ranges of values of \bar{Z}_2 for the 100 replications. The rest of the table entries can be interpreted similarly.*

Table 2: Maximal queue lengths and backlogs for various values of N

	N=10	N=100	N=1000	N=10000	N=100000
\bar{Z}_2	2 2.83 (2, 5)	7 5.08 (3, 9)	8 8.04 (5, 14)	10 10.52 (8, 14)	13 13.32 (11, 18)
\bar{Z}_3	2 3.07 (2, 6)	9 6.05 (4, 12)	11 9.15 (5, 14)	15 13.25 (9, 21)	16 16.78 (14, 21)
\bar{B}_2	2 2.38 (1, 5)	6 4.57 (2, 9)	11 7.54 (5, 14)	9 10.01 (8, 14)	13 12.91 (10, 17)
\bar{B}_5	1 1.73 (1, 5)	8 4.76 (2, 11)	9 7.73 (4, 13)	14 11.95 (8, 20)	15 15.44 (12, 20)

Figure 8 plots the values of the maximal queue lengths, against the problem size. It presents the results of the 100 replications of the simulation run. The digits indicate the number of replications with the same value, \times indicates a single replication and $+$ indicates more than 10 replications. Note the linear growth of the maximal queue length as a function of $\log_{10}(N)$.

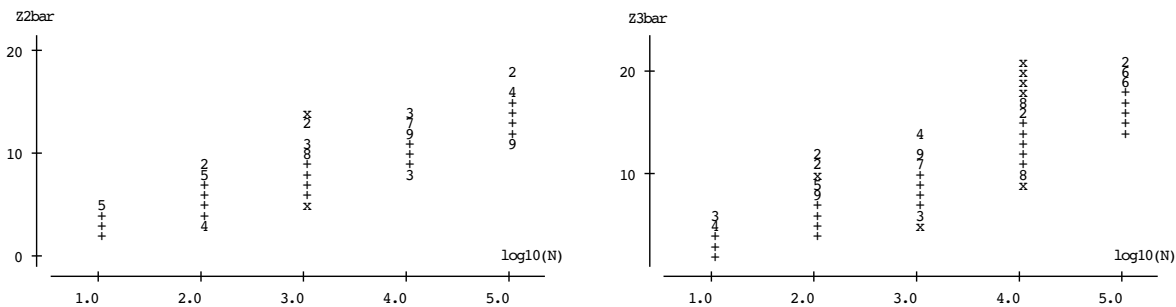


Figure 8: Maximal queue length plotted against $\log_{10}(N)$, for 100 replications

5 Concluding Remarks

Motivated also by an optimal fluid solution in Weiss [36] to the fluid model corresponding to a job-shop, Boudoukh, Penn and Weiss [6, 7] proposed a “greedy” algorithm for scheduling the job-shop. Their algorithm behaves like ours when the number of jobs in key buffers is sufficiently large (See Section 4 for a discussion of key buffers). A feature of their algorithm is that it constructs its own safety stocks automatically. They applied the algorithm to a

randomized version of the 10×10 job-shop problem. (Each route has N iid jobs.) The error bounds reported in their papers hardly grow as N gets large. Readers are referred to Tables 1 and 2 in [7] for more details.

Our paper has focused on providing an algorithm whose makespan is proven to be at most $O(\log N)$ away from optimality. As discussed in Section 4, choosing a right safety stock level requires a separate analysis, and there is no unique way to do it. This is the reason that we have not provided a numerical example to show how far our error bounds can be in practice. We view the algorithm in [6, 7] as a practical refinement of our theoretically proven algorithm. Thus, the examples in [6, 7] also support the good performance of our algorithm.

Theorem 1.1 is based on the assumption that $N_r = n_r N$ for each route r with $N \rightarrow \infty$. One would like to have an algorithm whose error bound is within $O(\log N)$ for arbitrary N_1, \dots, N_R with $N = N_1 + \dots + N_R \rightarrow \infty$. For identical jobs in each route, Bertsimas and Gamarnik [4] were able to prove an error bound of $O(\sqrt{N})$. We leave the general problem as a future research topic.

References

- [1] Adams, J., Balas, E. and Zawack, D. (1988) The shifting bottleneck procedure for job-shop scheduling. *Management Science* **34**, 391–401.
- [2] Anderson, E. J. (1981). A new continuous model for job-shop scheduling. *International J. Systems Science* **12**, 1469–1475.
- [3] Avram, F., Bertsimas, D. and Ricard, N. (1995). Fluid models of sequencing problems in open queueing networks: an optimal control approach. In F.P. Kelly and R. Williams, editors, *Stochastic Networks*, volume 71 of *IMA Volumes in Mathematics and its Applications*, pages 199–234, New York. Springer.
- [4] Bertsimas, D. and Gamarnik, D. (1999) Asymptotically optimal algorithms for job shop scheduling and packet routing. *J. of Algorithms* **33**, 296–318.
- [5] D.Barany, I. (1981) A vector sum theorem and its application to improving flow shop guarantees. *Mathematics of Operations Research* **6**, 445–452.
- [6] Boudoukh, T. (1999) Algorithms for solving job shop problems with identical and similar jobs, based on fluid approximation (Hebrew with English synopsis). M.Sc. Thesis, Technion, Haifa, Israel.
- [7] Boudoukh, T., Penn, M., Weiss, G. (2000) Scheduling jobshops with some identical or similar jobs. *J. of Scheduling*, to appear.
- [8] Carlier, J. and Pinson, E, (1988). An algorithm for solving the job-shop problem. *Management Science* **35**, 164–176.
- [9] Chan, L.M.A., Muriel, A. and Simchi-Levi, D. (1998) Parallel machine scheduling, linear programming, and parameter list scheduling heuristics. *Operations Research* **46**, 729–741.

- [10] Chen, H. and Yao, D. (1993) Dynamic scheduling of a multi class fluid network. *Operations Research* **41**, 104–1115.
- [11] Coffman, E.G. Jr. and G.S. Lueker. *Probabilistic Analysis of Packing and Partitioning Algorithms*. John Wiley, New York, 1992.
- [12] Connors, D., Feigin, G. and Yao, D. (1994) Scheduling semiconductor lines using a fluid network model. *IEEE Transactions on Robotics and Automation* **10**, 88–98.
- [13] Dai, J.G. 1995. On positive Harris recurrence of multi-class queueing networks: a unified approach via fluid limit models. *Annals of Applied Probability* **5**, 49–77.
- [14] Frenk, J.B.G., Rinnooy Kan, A.H.G. (1987) The asymptotic optimality of the LPT rule. *Mathematics Operations Research* **14**, 241–254.
- [15] Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.
- [16] Goldratt, E.M. and Cox, J. *The Goal: A Process of Ongoing Improvement*. 2nd edition North River Press, 1992.
- [17] Goldratt, E.M. *Critical Chain*. North River Press, 1997.
- [18] Harrison, J.M., (1996). The BIGSTEP approach to flow management in stochastic processing networks, *Stochastic Networks: Theory and Applications*, F. P. Kelly, I. Ziedins, S. Zachary, Editors, Oxford University Press.
- [19] Kaminsky, P. and Simchi-Levi, D. (1998) Probabilistic analysis and practical algorithms for the flow-shop weighted completion time problem. *Operation Research* **46**, 872–882.
- [20] Kumar, P.R. (1993) Re-entrant lines. *Queueing Systems Theory and Applications*, **13**, 87–110.
- [21] Kumar, P.R. Scheduling queueing networks: Stability, performance analysis and design. In F.P. Kelly and R. Williams, editors, *Stochastic Networks*, volume 71 of *IMA Volumes in Mathematics and its Applications*, New York, 1995. Springer.
- [22] Lann, A., Mosheiov, G. and Rinott Y (1998) Asymptotic optimality in probability of a heuristic schedule for open shops with job overlaps. *Operations Research Letters* **22**:63–68.
- [23] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. (1993) Sequencing and scheduling, algorithms and complexity. In *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*. edited by Graves, S.C., Rinnooy Kan, A.H.G. and Zipkin, P., North Holland.
- [24] Magalaras, C. (1999) Dynamic scheduling in multiclass queueing networks: stability under discrete-review policies. *Queueing Systems Theory and Applications* **31**, 171–206.

- [25] Magalaras, C. (2000) Discrete-review policies for scheduling stochastic networks: Trajectory tracking and fluid-scale asymptotic optimality. *Annals of Applied Probability*, **10**, 897–929.
- [26] Martin P. and Shmoys, D.B. (1996) A new approach to computing optimal schedules for the job-shop scheduling problem. *International IPCO Conference*, 389-403.
- [27] Meyn, S.P (1997) The policy improvement algorithm for Markov decision processes with general state space. *IEEE Transactions on Automatic Control*, **42**, 1663–1680.
- [28] Muth, J.F. and Thompson, G.L. (1954), *Industrial Scheduling*, Prentice Hall, New Jersey 225–251.
- [29] Pinedo, M. (1995) *Scheduling, Theory Algorithms and Systems*, Prentice Hall, New Jersey.
- [30] Ramudhin, A., Bartholdi, J.J., Calvin, J., Vande Vate, J.H., Weiss, G. 1996 A probabilistic analysis of 2-machine flow-shops. *Operations Research* **44**, 899–908.
- [31] Sevastyanov, S.V. (1987) Bounding algorithm for the routing problem with arbitrary paths and alternative servers. *Cybernetics* **22**, 773–781.
- [32] Sevastyanov, S.V. (1994) On some geometric methods in scheduling theory, a survey. *Discrete Applied Mathematics* **55**, 59–82.
- [33] Shwartz, A. and Weiss, A. *Large Deviations for Performance Analysis: Queues, Communications and Computing*. Chapman and Hall, New York, 1994.
- [34] Spaccamela, A.M., W.S. Rhee, L. Stoughie and S. van de Geer (1992). Probabilistic analysis of the minimum weighted flowtime scheduling problem. *Operations Research letters* **92**, 67–71.
- [35] Stolyar, A. L. (1995) On the stability of multiclass queueing networks: a relaxed sufficient condition via limiting fluid processes. *Markov Processes and Related Fields* **1**, 491–512.
- [36] Weiss, G. (1995) On optimal draining of fluid re-entrant lines. In *Stochastic Networks — IMA Volumes in Mathematics and its Applications*, Editors: F.P. Kelly and Ruth Williams, Springer-Verlag, New York, 93–105.
- [37] Weiss, G. (1999) Scheduling and control of manufacturing systems — a fluid approach *Proceedings of the 37 Allerton Conference, 21–24 September, 1999, Monticello, Illinois*, 557-586.
- [38] Williamson D.P., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevastyanov, S.V., Shmoys, D.B. (1997) Short shop scheduling. *Operations Research* **45**, 288–294.