J Sched (2010) 13: 509–529
DOI 10.1007/s10951-010-0174-0

A fluid approach to large volume job shop scheduling

Yoni Nazarathy · Gideon Weiss

Published online: 15 April 2010

Abstract We consider large volume job shop scheduling problems, in which there is a fixed number of machines, a bounded number of activities per job, and a large number of jobs. In large volume job shops it makes sense to solve a fluid problem and to schedule the jobs in such a way as to track the fluid solution. There have been several papers which used this idea to propose approximate solutions which are asymptotically optimal as the volume increases. We survey some of these results here. In most of these papers it is assumed that the problem consists of many identical copies of a fixed set of jobs. Our contribution in this paper is to extend the results to the far more general situation in which the many jobs are all different. We propose a very simple heuristic which can schedule such problems. We discuss asymptotic optimality of this heuristic, under a wide range of previously unexplored situations. We provide a software package to explore the performance of our policy, and present extensive computational evidence for its effectiveness.

Keywords Job shops · Makespan · Stochastic scheduling · Fluid approximation · Fluid tracking policy

Research supported in part by Israel Science Foundation Grant 249/02 and 454/05 and by European Network of Excellence Euro-NGI. Part of this research was conducted while Yoni Nazarathy was a student at the University of Haifa.

Y. Nazarathy
The Department of Mechanical Engineering and EURANDOM, Eindhoven University of Technology, Eindhoven, The Netherlands
e-mail: nazarathy@eurandom.tue.nl

G. Weiss (✉)
The Department of Statistics, The University of Haifa, Haifa, Israel
e-mail: gweiss@stat.haifa.ac.il

1 Introduction

Job shop scheduling has always been one of the most important and most difficult problems in scheduling theory. The formulation, which is well known (Lawler et al. 1993), is: jobs $n = 1, \ldots, N$ have to be scheduled on machines $i = 1, \ldots, I$. Job n requires r_n processing activities, moving through machines $\sigma(n,1), \ldots, \sigma(n,r_n)$ in that order, with processing times $x(n,1), \ldots, x(n,r_n)$. A schedule for such a problem is best described by a Gantt chart (Gantt 1910), as in Fig. 1. This includes in particular the completion times of the jobs, $C_n, n = 1, \ldots, N$. The utility of a schedule may be measured in many ways. The simplest and most common are the makespan, $\max_{n=1,\ldots,N} C_n$, the flowtime $\sum_{n=1,\ldots,N} C_n$, some other linear measure of the holding costs (weighted flowtime), or $\sum_{n=1,\ldots,N} h(C_n, D_n)$ where D_n are given due dates (number of late jobs, lateness, tardiness). In this paper we focus specifically on minimization of the makespan objective.

We are interested in job shops for which the number of machines I is fixed, and the number of activities of each job is bounded by a fixed $r_{\max} = \max\{r_1, \ldots, r_N\}$, but the number of jobs N is large. We call these *large volume job shops*. For these problems it is natural to adopt a fluid approach: solve a fluid approximation of the problem and use a schedule which tracks the fluid. Several papers (Bertsimas and Gamarnik 1999; Bertsimas and Sethuraman 2002; Bertsimas et al. 2003; Boudoukh et al. 1998, 2001; Dai and Weiss 2002) have considered this approach, suggested various tracking policies, and provided performance bounds. In most of these papers (with the exception of Dai and Weiss 2002) the following assumption is made: there is a fixed set of K jobs types, and a large number N of jobs which are all exact copies of these K job types. Each job of type k shares the same route and has the exact same processing times along this route as all the other copies.

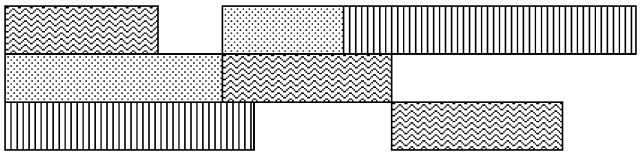

Fig. 1 Example of a jobshop schedule

In this paper we address a much more general large volume job shop, in which all N jobs are different, each with its own individual route and individual processing times. Thus we have a fixed set of I machines, and a fixed upper bound $r_{\max}$ on the route lengths, but apart from that, the N jobs are quite general. It is convenient in this case to classify all the possible activities of all the jobs into a fixed set of classes $k = 1, \ldots, K$. Activities of class k are performed at machine σ_k. The processing of job n can be described as moving through an ordered sequence of classes $k(n,1), \ldots, k(n,r_n)$, in which it is processed by machines $\sigma_{k(n,1)}, \ldots, \sigma_{k(n,r_n)}$. When in class k on step i of its route, job n will require a processing time $x(i,n)$ which can be any nonnegative real number. To be able to say something about these large volume job shops we make a probabilistic assumption on the processing times of jobs in activity class k: we assume that they are drawn from some population of processing times with probability distribution G_k, having finite mean m_k.

What we show in this paper is that the fluid approach works well for this general situation, and the results are quite similar to those obtained for the large volume job shop based on a fixed set of K jobs. In fact:

- The fluid solution can be calculated as before: the optimal fluid makespan is equal to the machine lower bound, and it can be achieved by a processor sharing policy which uses constant processing rates for each class and reduces the workload linearly as a function of time.
- The policy for tracking the fluid solution is very simple: it essentially uses the deviation of each class from the fluid solution to assign dynamic priority to that class.
- This policy is asymptotically optimal: as N increases the makespan increases linearly with N, while the suboptimality remains bounded by a quantity which is $o(N)$.
- Depending on several scenarios, the additive suboptimality bounds seem to be of order $O(1)$, $O(\log(N))$ or $O(\sqrt{N})$.

Our purpose in this paper is mainly expository: to explore how far the fluid approach can help in minimizing makespan under increasingly more general assumptions on the large volume job shop. Our emphasis is not on proving exact results, which are difficult to obtain in such general settings, but to try and understand how ideas which worked for the more restricted formulations need to be modified.

It turns out that the fluid solution is generalized easily, and also that tracking policies for the general case need not be more complicated; in fact, they may be simpler when the model is less structured. Furthermore they can be implemented with less information—the exact data of all the jobs is mostly not needed, and the schedule can be implemented online with very restricted information on the state of the system. Finally we wish to explore what price has to be paid for the increased generality: how large does the gap between the fluid lower bound solution and the actual makespan become, as we relax more assumptions?

To achieve this we concentrate mainly on simulation experimentation, to get some idea on the actual asymptotics of our large volume job shops. We provide a software package that can be used for this exploration and extensive analysis of simulation results.

The paper is structured as follows. In Sect. 2 we motivate the fluid approach to large volume job shops. We describe three types of job shop problems. The first of these is suitably handled by the use of combinatorial optimization. The second type of problem is suitably handled by the use of a stochastic Markov decision approach for steady state optimization of queuing networks. These two methodologies are unsuitable for the third type of problem, the large volume job shop, for which an approach that bridges these approaches is required. We survey in general terms some of the aspects of deterministic job shop scheduling and of stochastic optimization of queuing networks. We then outline our ideas of a combined approach, in which we solve a fluid job shop problem and schedule the jobs on line to track the fluid solution.

In Sect. 3 we present the fluid job shop and its solution, and prove that it provides a lower bound. This is a well-known result (Chen and Yao 2003; Weiss 1995), and we present it here mainly to illustrate how it works for our general large volume job shop. We then survey in Sect. 4 various previous papers including (Bertsimas and Gamarnik 1999; Bertsimas and Sethuraman 2002; Boudoukh et al. 2001; Dai and Weiss 2002), which used the fluid solution as a blueprint, and discuss those results.

In Sect. 5 we present our own method for tracking the fluid solution. We show that it generalizes the previous papers. In Sect. 6 we describe the implementation of our policy and a software system which we developed for that purpose. In Sect. 7 we present results of experiments which we carried out. We conclude in Sect. 8, with a discussion of fluid solutions in a wider context.

2 Three examples of job shops, and the approaches to their scheduling

To put our large volume job shop into context and motivate the fluid approach, we describe three typical scenarios.

2.1 Printed circuit board (PCB) assembly

A robot operates several tools in a sequence of steps to perform component assembly, insertion, placement, and attachment on a PCB. Here the sequence of activities for each tool is subject to various order constraints, and the whole process can often be formulated as a job shop scheduling problem, with the tools as machines, and components or subassemblies as the jobs (Kulak et al. 2007). The optimal schedule for this problem will then be coded and used by the robot repeatedly for each PCB. The inverse of the makespan will in this case be the throughput (the production rate) of the robot, and minimizing the makespan of the schedule will maximize the production of PCBs and the utilization of the robot.

The PCB scheduling problem is a pure scheduling problem, with a clear objective, which needs to be solved optimally once for each PCB design, and will then be reused often. It falls within the realm of combinatorial optimization problems.

2.2 Transmitting data through the Internet, control of data switches

Packets of data are transmitted from origin nodes to target nodes via intermediate nodes (cf. Stallings 2007). Here packets arrive and depart from the system, and the controller has no detailed information on processing times (packet sizes) or routes. The system is operating over a long time, under homogeneous conditions, and a choice of policy determines its steady state performance. There is no makespan in this problem, but the measure which is equivalent to minimal makespan in this infinite horizon problem is the throughput rate, the rate at which messages are transmitted along links of the network, and the rate at which messages are delivered at the various destinations. The control of the system is performed by allocating channel capacities to the packets, and by discretionary routing decisions. The equivalent of flowtime in this system is the sojourn time of packets in the system, or end-to-end delay. Often one needs to balance throughput with flowtime: higher throughput entails more congestion, and longer end-to-end delays.

Optimization of the steady state behavior of a communication network or of data switches is a problem of optimal control of a queuing network, and often falls within the realm of stochastic Markov decision processes.

2.3 Scheduling in a wafer fab

A wafer fab is a plant for the manufacturing of semiconductor computer chips (cf. Van Zant 2004). It may cost 3×10^9 \$, and will typically contain some 60,000 wafers in process at any moment in time, with work in process (WIP) sales value of 180×10^6 \$. Here each wafer follows a reentrant route of several hundred steps through some 60 machine groups. At any point in time one needs to sequence the work in front of each machine, which includes wafers in different stages of completion along the route. Initially the schedule starts with jobs in various stages (orders of wafers currently in production), and it needs to take into account future arriving jobs (future orders to be released into the system later). Furthermore, in this large system there will always be some noise, so that any schedule derived at a certain time may be subject to unforeseen changes in its execution at later times, including a few large changes and many small ones. Hence the schedule will need to be readjusted or recalculated as time moves on. Wafers may have due dates and may incur holding costs. By focusing on the makespan we will ignore these costs, but we will obtain high throughput and high utilization of bottleneck machines.

For this type of problem, the solution of the combinatorial optimization problem, which yields the optimal schedule for the current state of the job shop, is both intractable, because of the problem size, and of doubtful value, because of system noise. At the same time, trying to optimize the steady state of the wafer fab is both intractable and of little value because a wafer fab never reaches a steady state: by the time the initial load in the fab is exhausted (six weeks), the product mix, the processing steps, and the equipment will have undergone several changes. The wafer fab is an important example of a large volume job shop. It is this third type of problem which we wish to address in this paper.

Job shop scheduling by methods of combinatorial optimization, and control of multi-class queuing networks by optimization of the steady state are two areas of research which have been widely investigated. We give a brief survey of aspects relevant to this paper on both types of problems. We then outline our approach for the wafer fab type of problem, which forms a bridge between the two approaches.

2.4 Combinatorial optimization for job shop scheduling

Scheduling of a jobshop to minimize makespan is an NP-hard optimization problem (Garey and Johnson 1979; Lawler et al. 1993). Furthermore, finding a fully polynomial approximation scheme for this problem is NP-hard as shown by Williamson et al. (1997). In fact, it is shown in that paper that for a certain class of job shops, schedules with makespan of length 5 exist, but it is NP-complete to decide whether a schedule having a makespan of length 4 exists. This implies that calculation of an approximation which has performance ratio better than $5/4$ is NP-hard. This hard example of job shop problems is of a very special form: all the jobs consist either of two or three activities, and each activity is of unit length, while the number of machines is comparable to the number of jobs.

Taking a different approach, based on geometrical ideas expressed by the Steinitz Lemma, Sevastyanov (1987, 1994) (see also Barany 1981) describes a polynomial time algorithm, which achieves additive suboptimality, bounded by a quantity proportional to the duration of the longest activity.

A particular instance of job shop, the 10×10 job shop scheduling problem, became quite famous in the 1970s–1980s. The problem originated with Muth and Thompson (1954) and consists of 10 jobs, each with its own route through 10 machines: the routes as well as the processing times in this problem were randomly generated, and the problem was used as an illustration of a job shop in Conway et al. (1967). In the 1970s it was picked up as a challenge, and for more than 10 years the entire scheduling community was trying to find the optimal schedule: upper and lower bounds were posted and slowly converged until in 1987 Carlier and Pinson (1989) proved that the optimum makespan is 930. We will use this 10×10 example of a job shop as the main benchmark for our experimentation in this paper.

These results indicate that to solve real job shop scheduling problems one should look for heuristic methods. One should not expect them to provide guaranteed performance bounds for all problems, but one should tailor them to work well for particular types of problems: they should require a reasonable amount of computations, and provide a good solution with guaranteed error size for most problems that occur in practice.

A remarkably good heuristic, the shifting bottleneck heuristic, based on the work of Balas (1968), is described by Adams et al. (1988). This heuristic can solve the 10×10 problem in about 20 seconds, and it can solve some larger problems to optimality or obtain heuristic solutions which are close to the optimum, together with tight bounds, in reasonable time. The shifting bottleneck heuristic serves as the backbone in algorithms for a wide variety of variants of job shop problems. More recent approaches are discussed by Shmoys et al. (1994), Martin and Shmoys (1996), Sevastyanov and Woeginger (1998), and Jansen et al. (2000). See also the book of Pinedo (2002).

2.5 Steady state control of multi-class queuing networks

In a queuing network there are streams of arriving jobs of several types, requiring a sequence of processing activities from several servers. Jobs of the same type waiting for the same activity along their route define a class, and each server may serve queues of jobs of several classes, hence the name multi-class. The state of the system is described by the queue lengths of the various classes and by the residual processing and interarrival times. The policy which is used to control these networks is nonpredictive, since there is no exact data about the future. Furthermore, the initial state of the system and individual arrival and service times have little influence on the long-term behavior of the system, and the current state at each time may also not be observed in detail. The system is controlled based on probability distributions of the arrival streams and the processing times of each class. It is assumed that the system will run for a long time, and so under any homogeneous policy it will reach a steady state. Optimization of this steady state is the subject of stochastic dynamic programming, also named Markov decision processes. Optimal solutions and algorithms to compute them exist, but the computational complexity of these problems is even worse than the deterministic job shop scheduling problem. It is shown by Papadimitriou and Tsitsiklis (1999) that the general problem is EXP-complete: it cannot be solved in polynomial time.

Some important problems have priority type index solutions which are optimal, as discovered by Klimov and Gittins (1974, 1979), or nearly optimal. These include restless bandit problems introduced by Whittle (1981) and marginal productivity indexes introduced by Nio-Mora (2001, 2002, 2006).

Most often the optimal or near optimal solutions are much more complex, and there have been several approaches to construct heuristics to solve them. One technique is to solve the actual Markov decision problem approximately, by approximating the optimal value function; see, for example, Veatch (2005), and Moallemi et al. (2008).

Queuing networks in balanced heavy traffic can be approximated by diffusion processes, as suggested by Kushner (1989, 1990, 2001) and by Harrison (1988). On the diffusion scale input and output of the queues is approximated by Brownian motion, and the queues in heavy traffic are approximated by reflected Brownian motion. The limiting behavior of queuing networks in heavy traffic and their control has been the subject of much research, by Wein (1992), Harrison and Van Mieghem (1997), Tassiulas (1995), Dai and Lin (2005, 2006), Stolyar (2004), and others.

Other heuristics for obtaining approximations to optimal steady state control, based on both fluid and diffusion scaling, were suggested by Meyn (2001), Henderson et al. (2003), Chen et al. (2003). See also two recent books by Chen and Yao (2003) and by Meyn (2008).

2.6 Large volume job shops and their modeling by finite horizon multi-class queuing networks

Our approach to the scheduling of large volume job shops, such as the wafer fab, incorporates the features of the combinatorial job shop problem and the features of the queuing network steady state optimization problem. What is needed here is an approach which bridges the gap between these two types of problems. To do this we retain the dependence on the batch of jobs to be scheduled; we assume there is a finite set of N jobs to be scheduled. The number of jobs is

however large and involves a large total number of activities. Because there will be noise in this system, we do not wish to construct a schedule which will depend on the exact initially specified routes and processing times of all the jobs. Instead we wish to take advantage of the averaging that will occur in the large number of activities.

To that purpose we consider for each machine all the activities which are processed on that machine and divide them into a fixed number of classes. The classes of each machine i will be the constituency of this machine, denoted by $k \in \mathcal{K}_i$. We then base our schedule not on the processing times of activities of the individual jobs, but on the population of processing times of the activities of the entire class. This yields a probability distribution of processing times, G_k for each class k. Similarly, we do not base our schedule on the routes of the individual jobs: instead we consider all the routing steps which follow the completion of the activities of class k. This yields routing probabilities $P_{kk'}$ where a fraction $P_{kk'}$ of activities of class k are followed by an activity of type k', and where a fraction $1 - \sum_{k'} P_{kk'}$ of the jobs complete their processing and leave the system after an activity of class k.

This is exactly the model of a multi-class queuing network (cf. Harrison 1988), in which there are I machines and K classes. Jobs of class k are served according to service time distribution G_k, and are then routed to class k' with probability $P_{kk'}$ or leave the system with probability $1 - \sum_{k'} P_{kk'}$. Here the queue length $Q_k(t)$ denotes the number of jobs of type k in the system at time t.

To describe the large volume job shop, we specify as $Q_k(0)$ the initial number of jobs of type k, for $k = 1, \ldots, K$, so that $N = \sum_k Q_k(0)$. Since we are only interested in scheduling this finite batch of jobs, there are no exogenous arrivals in this multi-class queuing network, and we only control it until the initial jobs are completed. We have thus modeled our large volume job shop as what we will call *a finite horizon multi-class queuing network*.

Note: One can include release times in this finite horizon multi-class queuing network model, by letting all the jobs be initially in class $k = 0$, with its own dedicated machine 0 which releases the jobs into the system.

Evaluation of the performance Combinatorial optimization involves a finite number of possible schedules with a unique verifiable optimum. It is then possible to evaluate exactly the performance of a policy for each instance. It is the accepted practice to evaluate the heuristic on the basis of its worst-case performance: the performance for the worst problem instance. This worst-case performance guarantee may often not be representative of the performance of the heuristic on standard problems. In contrast, queuing models which are stochastic require further modeling assumptions: it is assumed that the problem arises from a limited population of problems, which is described by a probability distribution, and one then performs an average case analysis for this population of problems, by evaluating the expected performance. If the probabilistic assumptions are reasonable, the performance evaluation will indeed reflect the actual performance.

For a large volume job shop one would in practice have a well-defined population of jobs, routes, and processing times, and the distribution of class processing times and the routing probabilities may be quite realistic. We therefore take the approach of evaluating the expected performance, under probabilistic assumptions.

3 The fluid solution

Consider the example in Fig. 1, and assume that we can divide each job into two half-sized jobs, as described in Fig. 2. Using the half-sized blocks, subject to the original routes, we can repeat the schedule of Fig. 1 twice, as done on the top of the figure. However, we can now do better—we are now able to schedule all the work without any idle time on any of the machines, as seen on the bottom part of the figure. It is important to note that this half job representation allows us more than just the preemption of each activity in the middle. It actually allows us to start the next half activity on the route immediately when a half activity is finished.

For any job shop scheduling problem the total processing time of all the jobs on each one of the machines imposes a lower bound on the makespan; this is known as the machine lower bound. Also, the total duration of the whole route of each job imposes a lower bound on the makespan, known as the job lower bound. With many jobs we expect the machine lower bound to be much larger than the job lower bound. In Fig. 1 the makespan is larger than either of the lower bounds, but with the two repeated half jobs in Fig. 2 the machine lower bound is achieved, and the job lower bound, which is now half of its original value, has become almost irrelevant.

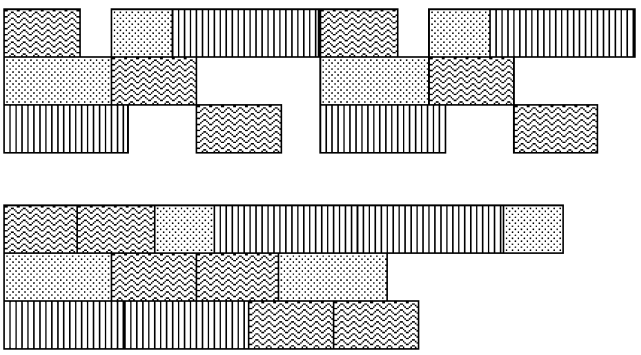

Fig. 2 Improved makespan for two half-sized job shops

If we were to look at any job shop, and divide each of the jobs into N jobs with $1/N$ processing times, then the job lower bound would become very small. At the same time, we would guess that similar to Fig. 2, there will be very little idleness, and the makespan can near the machine lower bound.

The fluid problem takes this process of divisibility to the limit, and indeed in the fluid problem the makespan equals the machine lower bound, and there exists a very simple policy which achieves it (it is far from unique). We describe now the fluid job shop, and present this result as a theorem. The result is well known, and various versions of this theorem were proved in the past, starting with Weiss (1995), where it was first shown for a re-entrant line. It was proved again in the context of job shop scheduling in Bertsimas and Gamarnik (1998, 1999, 2001), and a proof in the context of multi-class queuing networks is given in the book of Chen and Yao (2003) §12.3.2. For completeness and for clear exposition of the fluid ideas, we present the proof again here, in a version that is suitable for a general large volume job shop that is modeled as a finite horizon multi-class queuing network.

Intuitively, imagine that we dismantle each job into tiny job molecule particles, each of which can move through the route of the job on its own. We assume head-of-the-line priority among the particles of each job, so that in each activity the particles of each job will arrive and depart starting from the first particle, in their original order. We can now imagine each machine to be serving all the particles that it has waiting in a round robin fashion. Then the inventory of each activity of each job will decrease linearly, simultaneously with inventories of other jobs sharing the same class. Particles will then move on to their next class out of each machine in fixed proportions, so that the inflow and outflow of each of the classes in each of the machines will be at a linear rate. By adjusting the round robin rates for each job in each class, the inventories at each machine will reduce linearly to zero exactly in the time determined by the machine lower bound of the bottleneck machine. Note that classes which start off with zero inventories will have particles moving in and out at equal rates, and will stay with negligible inventory throughout.

We now formalize this intuitive discussion. To do so we look at our job shop scheduling problem as a finite horizon queuing network in which the jobs are classified into classes $k = 1, \ldots, K$. We let $I_k(n, j)$ be the indicator (value 1 for yes, 0 for no) that the processing activity of step j of job n is a class k activity. We write the initial number of jobs of class k in the job shop:

$$Q_k(0) = \sum_{n=1}^{N} I_k(n, 1).$$

We also write the total number of class k activities, included all the steps of all the jobs:

$$Q_k^+(0) = \sum_{n=1}^{N} \sum_{j=1}^{r_n} I_k(n, j),$$

and the total number of route transitions from class k to class k':

$$\Phi_{kk'}^+ = \sum_{n=1}^{N} \sum_{j=1}^{r_n - 1} I_k(n, j) I_{k'}(n, j+1).$$

We now describe the dynamics of the job shop. For $k \in \mathcal{K}_i$ denote by $U_k(t)$ the total time which machine i allocates to the processing of jobs of class k. Let $S_k(U_k(t))$ count the number of jobs which have completed their processing at class k after a total processing time of $U_k(t)$, and have left class k by time t, and let $\Phi_{kk'}(S_k(U_k(t)))$ count how many of these jobs were routed to class k' on their next processing step. Then $Q_k(t)$, the number of jobs in class k at time t (queue k), is

$$Q_k(t) = Q_k(0) - S_k\big(U_k(t)\big) + \sum_{k'} \Phi_{k'k}\big(S_{k'}\big(U_{k'}(t)\big)\big), \qquad (1)$$

and the total number of activities of class k which at time t have not yet completed processing is

$$Q_k^+(t) = Q_k^+(0) - S_k\big(U_k(t)\big). \qquad (2)$$

This is a complete description of the dynamics of the job shop in terms of the finite horizon queuing network, but it is subject to several obvious as well as hidden additional constraints. In particular, queues have a nonnegative integer number of jobs in them, time can be allocated only to queues with ≥ 1 jobs, and time allocation starts at 0 at time zero, is continuous nondecreasing and can only increase at a rate ≤ 1. This means that $U_k(t)$ are Lipschitz, with constant 1, and therefore have a derivative defined almost everywhere. We denote the derivative by $u_k(t)$ wherever it is defined, so that $u_k(t) = \dot{U}_k(t) = \frac{d}{dt} U_k(t)$ and $U_k(t) = \int_0^t u_k(s)\, ds$. In fact, for a job shop schedule the only possible time allocation rates are $u_k(t) \in \{0, 1\}$, and $u_k(t)$ can only change from 1 to 0 when $S_k(U_k(t))$ increases, i.e., when a job in class k completes its processing. Writing some of these requirements, we get constraints which supplement (1), (2),

$$\begin{aligned}
&Q_k(t) \quad \text{is integer and } \geq 0,\\
&U_k(0) = 0, \qquad u_k(t) \in \{0, 1\}, \qquad \sum_{k \in \mathcal{K}_i} u_k(t) \leq 1,\\
&\text{if } Q_k(t) = 0 \quad \text{then } u_k(t) = 0,\\
&u_k(t) \quad \text{can decrease only when } S_k\big(U_k(t)\big) \text{ increases.}
\end{aligned} \qquad (3)$$

The fluid model of the job shop problem relaxes these constraints: we now assume that the jobs are composed of fluid, so that it is no longer required that the queue lengths be integer. We also assume that the machines are infinitely divisible and can allocate processing at rates which are real numbers between 0 and 1. We also assume that all jobs of a class are processed at a uniform rate, and are routed in fixed proportions.

To obtain the constant processing rate we calculate the total amount of processing required for class k:

$$T_k = \sum_{n=1}^{N} \sum_{j=1}^{r_n} I_k(n,j) x(n,j).$$

The average processing time m_k and average processing rate $\mu_k = 1/m_k$ for class k activities and the routing fraction $P_{kk'}$ from class k to k' are given by

$$m_k = T_k / Q_k^+(0), \qquad \mu_k = Q_k^+(0)/T_k,$$
$$P_{kk'} = \Phi_{kk'}^+ / Q_k^+(0).$$

With these quantities we can now write the relaxed constraints for the fluid network. Denoting by $q_k(t), q_k^+(t)$ the immediate and the total class k fluid in the system at time t, we have the fluid dynamics:

$$q_k(t) = q_k(0) - \mu_k \int_0^t u_k(s)\,ds + \sum_{k'} P_{k'k} \mu_{k'} \int_0^t u_{k'}(s)\,ds, \tag{4}$$

$$q_k^+(t) = q_k^+(0) - \mu_k \int_0^t u_k(s)\,ds, \tag{5}$$

where the initial conditions are $q_k(0) = Q_k(0)$, $q_k^+(0) = Q_k^+(0)$.

In matrix notation we define the constituency matrix C with elements $c_{i,k} = 1$ if $k \in \mathcal{K}_i$, $c_{i,k} = 0$ otherwise, and the input-output matrix R defined by

$$R = \left(I - P^T\right) \text{diagonal}(\mu)$$

where I is the unit matrix, ^{T} denotes transpose, and diagonal (μ) is the diagonal matrix of processing rates. We also let $\mathbf{1}$ denote a vector of 1's, and we let $1_{q(t)\neq 0}$ be the indicator of the event that not all the fluid queues are empty.

We then get the fluid minimum makespan problem:

$$\begin{aligned} \min \quad & \int_0^\infty 1_{q(t)\neq 0}\,dt \\ \text{s.t.} \quad & q(t) = q(0) - \int_0^t Ru(s)\,ds, \\ & Cu(t) \le \mathbf{1}, \\ & u(t), q(t) \ge 0. \end{aligned} \tag{6}$$

Theorem 3.1 *Let $\mathbf{T}_i = \sum_{k\in\mathcal{K}_i} T_k$ be the total workload of machine i, and let $i^* \in \arg\max\{\mathbf{T}_1, \ldots, \mathbf{T}_I\}$ be a bottleneck machine (it may be unique or there may be several), and $\mathbf{T}^* = \max\{\mathbf{T}_1, \ldots, \mathbf{T}_I\}$ be the machine lower bound for the job shop. Then the fluid problem (6) achieves as optimal objective value the machine lower bound $\mathbf{T}^*$. Optimal machine allocation rates and optimal fluid levels (nonunique) for the various classes are given, for $0 < t < \mathbf{T}^*$, by*

$$\begin{aligned} u_k(t) &= \frac{q_k^+(0)}{\mu_k \mathbf{T}^*}, \\ q_k(t) &= q_k(0)\left(1 - \frac{t}{\mathbf{T}^*}\right), \\ q_k^+(t) &= q_k^+(0)\left(1 - \frac{t}{\mathbf{T}^*}\right). \end{aligned} \tag{7}$$

Proof Clearly, the value of $\mathbf{T}^*$ is a lower bound for the fluid problem. Hence all we need to show is that the solution (7) satisfies the constraints, and reaches this objective value.

Calculating $Cu(t)$, for each i, we have $\sum_{k\in\mathcal{K}_i} u_k(t) = \sum_{k\in\mathcal{K}_i} \frac{q_k^+(0)}{\mu_k \mathbf{T}^*} = \frac{\mathbf{T}_i}{\mathbf{T}^*} \le 1$. Clearly, also $u(t) \ge 0$.

Calculating $q_k^+(t)$, for each k, for these machine allocation rates:

$$\begin{aligned} q_k^+(t) &= q_k^+(0) - \mu_k \int_0^t u_k(s)\,ds \\ &= q_k^+(0) - \mu_k \frac{q_k^+(0) t}{\mu_k \mathbf{T}^*} = q_k^+(0)\left(1 - \frac{t}{\mathbf{T}^*}\right). \end{aligned}$$

Calculating $q_k(t)$, for each k, for these machine allocation rates:

$$\begin{aligned} q_k(t) &= q_k(0) - \mu_k \int_0^t u_k(s)\,ds + \sum_{k'} P_{k'k} \mu_{k'} \int_0^t u_{k'}(s)\,ds \\ &= q_k(0) - \frac{t}{\mathbf{T}^*}\left(q_k^+(0) - \sum_{k'} P_{k'k} q_{k'}^+(0)\right) \\ &= q_k(0) - \frac{t}{\mathbf{T}^*}\left(q_k^+(0) - \sum_{k'} \Phi_{k'k}\right) \\ &= q_k(0) - \frac{t}{\mathbf{T}^*}\left(\sum_{n=1}^{N}\sum_{j=1}^{r_n} I_k(n,j) \right. \\ &\quad \left. - \sum_{k'}\sum_{n=1}^{N}\sum_{j=1}^{r_n-1} I_{k'}(n,j) I_k(n,j+1)\right) \\ &= q_k(0)\left(1 - \frac{t}{\mathbf{T}^*}\right) - \frac{t}{\mathbf{T}^*}\left(\sum_{n=1}^{N}\sum_{j=2}^{r_n} I_k(n,j) \right. \\ &\quad \left. - \sum_{n=1}^{N}\sum_{j=1}^{r_n-1} I_k(n,j+1) \sum_{k'} I_{k'}(n,j)\right) \\ &= q_k(0)\left(1 - \frac{t}{\mathbf{T}^*}\right). \end{aligned}$$

Clearly, for all $0 < t < \mathbf{T}^*$, $q_k(t) \geq 0$, and furthermore, $q_k(\mathbf{T}^*) = 0$ for all k. Hence the objective for this solution equals $\mathbf{T}^*$. □

Note By using constant machine allocations the fluid levels in each class decrease linearly from the initial value to 0 by time $\mathbf{T}^*$. Any class which did not have any fluid initially will remain at a level $q_k(t) = 0$ throughout. This will be achieved by having equal rates of inflow and outflow for such classes. In this solution every bottleneck machine is working at full utilization. Machines which are not bottlenecks work at a constant less than full utilization, the utilization is $T_i/\mathbf{T}^*$.

3.1 Example: The ten by ten job shop scheduling problem (Muth and Thompson 1954)

The data for this example is given in Table 1. There are 10 machines and 10 jobs, the routes are permutations of the machines, so each route visits all the machines exactly once. Viewed as a finite horizon queuing network, there are 100 activities, so there are 100 classes. We shall denote class k by the pair (n, j) which is step j of job n. The cell of row n column j of the table refers to class (n, j), with $\sigma(n, j)$ as the top entry, and $x(n, j)$ as the bottom entry. The total work for each job is given in the right margin column of the table. The job lower bound is for job 4, and equals 655.

Class $k = (n, j)$ is in the constituency of machine $\sigma(n, j)$, and it has a single activity that goes through this class. Hence, $q_{(n,1)} = 1, q_{(n,j)} = 0$, $j = 2, \ldots, 10$, and $q^+_{(n,j)} = 1$ for all (n, j). The processing duration for class k is simply $T_k = x(n, j)$. The machine workloads $\mathbf{T}_i$, $i = 1, \ldots, 10$ are given at the bottom of Table 1. The bottleneck machine is machine 4, with $\mathbf{T}^* = 631$.

The fluid solution is very simple. The machine allocation to class $k = (n, j)$ is simply $u_{(n,j)}(t) = \frac{x(n,j)}{\mathbf{T}^*}$. The class buffer content is $q_{(n,1)}(t) = (1 - \frac{t}{\mathbf{T}^*})$, while for $j = 2, \ldots, 10$ $q_{(n,j)}(t) = 0$. For each job n, fluid flows out of the class of step 1 of the job, at rate $\frac{t}{\mathbf{T}^*}$, through all the class buffers of the other steps (which have an input rate equal to the output rate and hence stay empty), and out of the last step. The system is empty at $\mathbf{T}^* = 631$. Figure 3 describes the fluid picture. It is this picture which we wish to track for large volume job shops.

Table 1 The 10×10 job shop scheduling problem

Job	Step										
	1	2	3	4	5	6	7	8	9	10	Total
1	1	2	3	4	5	6	7	8	9	10	
	29	78	9	36	49	11	62	56	44	21	395
2	1	3	5	10	4	2	7	6	8	9	
	43	90	75	11	69	28	46	46	72	30	510
3	2	1	4	3	9	6	8	7	10	5	
	91	85	39	74	90	10	12	89	45	33	568
4	2	3	1	5	7	9	8	4	10	6	
	81	95	71	99	9	52	85	98	22	43	655
5	3	1	2	6	4	5	9	8	10	7	
	14	6	22	61	26	69	21	49	72	53	393
6	3	2	6	4	9	10	1	7	5	8	
	84	2	52	95	48	72	47	65	6	25	496
7	2	1	4	3	7	6	10	9	8	5	
	46	37	61	13	32	21	32	89	30	55	416
8	3	1	2	6	5	7	9	10	8	4	
	31	86	46	74	32	88	19	48	36	79	539
9	1	2	4	6	3	10	7	8	5	9	
	76	69	76	51	85	11	40	89	26	74	597
10	2	1	3	7	9	10	6	4	5	8	
	85	13	81	7	64	76	47	52	90	45	560

Machine workloads for the 10×10 *problem*

Machine i	1	2	3	4*	5	6	7	8	9	10
$\mathbf{T}_i$	493	548	576	631*	534	416	491	499	531	410

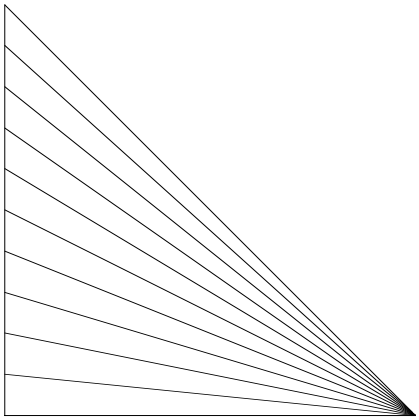

Fig. 3 The form of the fluid solution of the 10×10 example

4 Approaching the machine lower bound

In the previous section it was established that the fluid solution for a job shop scheduling problem has the machine lower bound as its makespan. Intuition then suggests that the minimum makespan of a large volume job shop should be close to this lower bound. This is in fact true in various situations, and furthermore, heuristics which come close to the optimum and to the lower bound can be developed for these situations. In this section we survey various papers that discuss heuristics which indeed achieve this lower bound asymptotically, as the volume of the job shop increases.

The various heuristics differ in three ways: in the setup, i.e., the type of large volume job shop that is considered, in the scheduling algorithm, and in the degree to which they approach the lower bound. The following table thumbnails some of these features of the various papers. More details and further explanations are given in the detailed survey that follows in Table 2.

4.1 The synchronized lots algorithm of Bertsimas and Gamarnik (1999)

Bertsimas and Gamarnik (1999) consider the following setup for their large volume job shop problem: there is a fixed number of machines numbered $i = 1, \ldots, I$, and a fixed set of job types, numbered $m = 1, \ldots, M$, each with its route $\sigma(m, 1), \ldots, \sigma(m, r_m)$ and processing times $x(m, 1), \ldots, x(m, r_m)$. We use $\mathcal{K} = \{k : k = (m, j),\ j = 1, \ldots, r_m,\ m = 1, \ldots, M\}$ to denote the set of activities for all these jobs, and $\mathcal{K}_i = \{k : \sigma(k) = i\}$ to denote the constituency of machine i. Let $r_{\max} = \max\{r_1, \ldots, r_M\}$ denote the length of the longest route. Let $y_i = \sum_{k \in \mathcal{K}_i} x(k)$ and $y_{\max} = \max\{y_1, \ldots, y_I\}$ denote the individual machine work requirements and the maximal machine lower bound for performing one repetition of this set of jobs. Then $I, M, r_{\max}, y_{\max}$ quantify the job shop setup.

For the large volume job shop with this setup Bertsimas and Gamarnik assume that there are N_m jobs of type m to be scheduled, with a total number of jobs $N = \sum_{m=1}^{M} N_m$. From this one can calculate the fluid solution with makespan (that equals the machine lower bound) T^*.

Bertsimas and Gamarnik propose the following algorithm which we refer to here as a synchronized lots algorithm: choose a number Ω, to be determined soon. Determine a lot size for each type of job: $a_m = \lceil \frac{N_m \Omega}{T^*} \rceil$ (here $\lceil \cdot \rceil$ denotes rounding upwards). Perform a sequence of lot schedules, $\ell = 1, 2, \ldots$, each of which lasts for a duration of $\Omega + y_{\max}$. For lot ℓ at time $(\ell - 1)(\Omega + y_{\max})$, assign activities to each machine as follows: at machine i for job type m, and activity $k = (m, j) \in \mathcal{K}_i$, consider for processing only such jobs which are available for processing of activity k at that time, and choose for processing all of them or a_m of them, whichever is smaller. This is gated service with an upper bound: only jobs available at the machine at the start of a lot-schedule will be processed and not any later arrivals, and no more than a_m activities of class $(m, j) \in \mathcal{K}_i$

Table 2 Heuristics for large volume job shops using the fluid approach

Job shop setup	Type of algorithm	Optimality gap	Reference
$\frac{N}{M}$ copies of a set of M fixed jobs	Synchronized cyclic pipeline	$O(1)$	Boudoukh et al. (1998), Bertsimas and Gamarnik (1999)
N_m copies of fixed $m = 1, \ldots, M$ jobs	Synchronized $O(\sqrt{N})$ lots	$O(\sqrt{N})$	Bertsimas and Gamarnik (1999)
N_m copies of fixed $m = 1, \ldots, M$ jobs	Greedy tracking	$O(1)$	Bertsimas and Sethuraman (2002)
N_m random jobs on routes $m = 1, \ldots, M$	Synchronized cyclic pipeline with safety stocks	$O(\log N)$	Dai and Weiss (2002)
N random jobs	Greedy tracking	Between $O(1)$ and $O(\sqrt{N})$	Indicated by simulation here, and by Nazarathy and Weiss (2009)

will be processed. Perform the processing of the lot at each machine and then idle. Clearly, each machine can process its lot continuously with no idling until it is complete. It is seen immediately that each machine will complete its lot in a duration which is $\leq \Omega + y_{\max}$. It will then idle until the end of the ℓ period, at $\ell(\Omega + y_{\max})$.

This processing of gated lots with bounded lot size produces synchronization between the machines: in the first lot a_m jobs of type m are available to receive the first processing step of their route. After processing of the first lot, these jobs are available for scheduling of the activity of the second step on their route, so at the beginning of the second lot, the first and second steps on each route will have a_m jobs available. After $r_{\max}$ lots all the machines will be synchronized: each machine i will perform exactly a_m activity processing steps for each activity $k = (m, j) \in \mathcal{K}$, for as long as there are any such activities left in the job shop. All the first step activities of all the N_m jobs of type m will then be completed after $\lceil N_m/a_m \rceil$ lot-schedules. By definition of a_m, $\lceil N_m/a_m \rceil \leq \frac{T^*}{\Omega}$. Hence, all the first step activities of all the jobs on all the routes will be completed after no more than $\lceil \frac{T^*}{\Omega} \rceil$ lot-schedules. Because of the synchronization, the last activities of all the jobs on all the routes will be completed by $r_{\max} - 1$ periods later. So the synchronization heuristic will achieve a makespan T^{Lots} which is bounded by

$$T^{\text{Lots}} \leq \left(\frac{T^*}{\Omega} + r_{\max}\right)(\Omega + y_{\max})$$
$$= T^* + r_{\max}\Omega + \frac{T^* y_{\max}}{\Omega} + r_{\max} y_{\max}.$$

This is minimized (similar to EOQ lot sizing in Inventory theory), by choosing $\Omega = \sqrt{T^* y_{\max}/r_{\max}}$, yielding

$$T^* \leq T^{\text{Opt}} \leq T^{\text{Lots}} \leq T^* + 2\sqrt{T^* r_{\max} y_{\max}} + r_{\max} y_{\max}.$$

As can be seen, the synchronized lots-schedule is asymptotically optimal as $N \to \infty$, with an additive optimality gap of order $O(\sqrt{N})$.

4.2 The cyclic pipeline schedules of Bertsimas and Gamarnik (1999) and Boudoukh et al. (1998, 2001)

Bertsimas and Gamarnik (1999) and Boudoukh et al. (1998, 2001) consider a different setup, which is more restrictive, but achieves a tighter bound. Once again they assume a fixed set of machines and a fixed set of jobs, as above, and a set of fixed integers b_m, $m = 1, \ldots, M$. The number of jobs of type m is now restricted to be $N_m = b_m \tilde{N}$. $\tilde{N}$ is called the multiplicity of the job shop scheduling problem. The total number of jobs is then $N = \tilde{N} \sum_m b_m$

One can now compute T^* for any number of repetitions $\tilde{N}$, and define $\Omega = T^*/\tilde{N}$. One then modifies the synchronization algorithm somewhat: the lot sizes are determined by b_m instead of a_m, and the processing duration of each lot is simply Ω (instead of $\Omega + y_{\max}$).

The job shop schedule will be completed in $\tilde{N} + r_{\max} - 1$ lots, and so the makespan of this cyclic heuristic T^{Cyclic} will satisfy

$$T^* \leq T^{\text{Opt}} \leq T^{\text{Cyclic}} \leq T^* + \Omega(r_{\max} - 1).$$

Note that $\Omega(r_{\max} - 1)$ is a fixed quantity, independent of $\tilde{N}$, while T^* grows linearly with $\tilde{N}$. This synchronized cyclic schedule is asymptotically optimal as $N \to \infty$, with an additive optimality gap of order $O(1)$.

We may without loss of generality consider only the case of $b_m = 1$, by considering b_m copies of job of type m as if they are different types. There are now M types (corresponding to the sum of the original b_ms), with $\tilde{N}$ repetitions of each. This case was examined more closely by Boudoukh et al. (1998, 2001).

It is now seen that the schedule is indeed cyclic: there is an initial synchronization phase, which lasts at most $r_{\max} - 1$ lots. This is the buildup part of the schedule. After that the schedule of each lot includes the processing of exactly one activity of each class $k = (m, j)$, and this continues until the $\tilde{N}$th lot is completed. This is the cyclic part of the schedule. Following lot $\tilde{N}$ one has again at most $r_{\max} - 1$ incomplete lots until the full schedule is completed. This is the runout part of the schedule.

One can see in this cyclic schedule the phenomena of pipelining. Clearly, since only jobs which are available at the beginning of the cycle are scheduled in this cycle, all the activities which are processed within a cycle belong to different jobs: the pipeline holds at any time one job for each of the different $k = (m, j)$ activities, and these all move down the pipeline one step at a time. Cyclic schedules have also been considered previously, for various flow shop scheduling problems, in Hanen (1994), Hochbaum and Shamir (1991), Matsuo (1990), McCormick et al. (1989), Roundy (1992).

4.3 The greedy fluid tracking algorithm of Bertsimas and Sethuraman (2002)

Bertsimas and Sethuraman (2002) consider the same setup as Bertsimas and Gamarnik (1999) in Sect. 4.1, with N_m copies of jobs (the types) $m = 1, \ldots, M$. They improve on the results of Bertsimas and Gamarnik (1999) by achieving an additive upper bound which is $O(1)$, similar to the bound for the cyclic schedule of Sect. 4.2, in which the numbers of jobs of each type were equal.

The idea is as follows. The fluid solution can be viewed as a processor sharing policy, and the cyclic schedule of Bertsimas and Gamarnik (1999), Boudoukh et al. (1998, 2001) in Sect. 4.2 can be viewed as a round robin approximation to this processor scheduling schedule. When the number of jobs of each type is unequal, round robin no longer works. To approximate the fluid solution, Bertsimas and Gamarnik

(1999) in the synchronized lot algorithm have divided all the activities into lots of equal size, and approximated the fluid solution by performing a round robin of lots. However, the size of the lots $O(\sqrt{N})$ led to an additive gap of $O(\sqrt{N})$.

The problem of approximating the performance of processor sharing by processing discrete individual tasks without machine splitting or preeemption is frequently encountered in computing and communications, and several simple and elegant methods have been proposed when round robin does not work (see Demers et al. 1990; Greenberg and Madras 1992; Parekh and Gallager 1993, 1994; Zhang 1995). Bertsimas and Sethuraman adapt one of these methods, the *fair queuing based on start times* (FQS) algorithm.

The algorithm works as follows: number the N_m jobs of type m by $n = 1, \ldots, N_m$. Each of these jobs will need to undergo processing activities $(m, 1), \ldots, (m, r_m)$, of durations $x(m, 1), \ldots, x(m, r_m)$. Define the following.

Discrete start time $DS_{m,j}(n) =$ the actual time at which the nth processing of activity (m, j) starts.

Fluid start time

$$FS_{m,j}(1) = 0,$$

$$FS_{m,j}(n) = FS_{m,j}(n-1) + \frac{T^*}{N_m}, \quad n > 1.$$

Nominal start time

$$NS_{m,1}(n) = FS_{m,1}(n),$$

$$NS_{m,j}(1) = NS_{m,j-1}(1) + x_{(i,k)}, \quad j > 1,$$

$$NS_{m,j}(n) = \max\left\{NS_{m,j}(n-1) + \frac{T^*}{N_m},\ DS_{m,j-1}(n)\right\},$$

$n, j > 1$.

The greedy fluid tracking algorithm is extremely simple: whenever a machine becomes free, or whenever a job arrives at an idle machine, this machine will start processing the job with the lowest nominal start time *NS*.

Bertsimas and Sethuraman (2002) show first that all available jobs do indeed possess a nominal start time, so that the algorithm can be executed, and they show that the makespan $T^{\text{Greedy Tracking}}$ has the following worst-case performance bounds:

$$T^* \le T^{\text{Opt}} \le T^{\text{Greedy Tracking}} \le T^* + \max_{i=1,\ldots,I}(2z_i + y_i),$$

where the quantities z_i, y_i for machine i are derived from processing of a single lot of the M jobs,

$$z_i = \max\{x_{m,j} : \sigma(m, j) = i\},$$

$$y_i = \sum_{(m,j):\sigma(m,j)=i} x_{m,j}.$$

The greedy tracking algorithm is of course asymptotically optimal as $N \to \infty$, with an additive performance gap $O(1)$.

An important feature of the greedy tracking algorithm is that, while the nominal start times track the fluid solution (i.e., they are close to the fluid solution), the actual schedule (i.e., the actual start times) need not track the fluid solution: jobs may be scheduled well ahead of their nominal start times. For example, if all the activities which a machine is processing are first step activities of several types of jobs, and if this machine is not a bottleneck machine, then it will work with no idling until it completes all its work at the time T_i, well ahead of T^*. It is for this reason that we refer to this algorithm as greedy tracking.

4.4 Scheduling similar but nonidentical jobs in a cyclic schedule

In all the papers which we surveyed so far the job shops had a finite set of fixed processing times, $x(m, j)$, $j = 1, \ldots, r_m$, $m = 1, \ldots, M$, and all the jobs in the large volume job shop had exactly these processing times. This is a highly unrealistic assumption. We now describe the work of Dai and Weiss (2002) in which processing times are not restricted.

Dai and Weiss (2002) consider a setup similar to Bertsimas and Gamarnik (1999), Boudoukh et al. (1998, 2001) described in Sect. 4.2: there are $m = 1, \ldots, M$ job types, with fixed routes $(m, 1), \ldots, (m, r_m)$, and $\tilde{N}$ jobs of each type. The difference is that now the processing times of the $\tilde{N}$ jobs of route m are not identical. We now have processing times $X_{m,j}(n)$ for step j of job n of type m. Dai and Weiss model these processing times as drawn independently from some processing time distributions with means $x(m, j)$.

They suggest a schedule which is similar to the cyclic schedule of Bertsimas and Gamarnik (1999), Boudoukh et al. (1998, 2001) described in Sect. 4.2, which they name a safety stock offset cyclic schedule. We describe here a modified version of this schedule—this modification makes the schedule more natural, without essentially changing the analysis and results presented in Dai and Weiss (2002).

Locate the bottleneck machine based on the average processing times, $x(m, j)$, by calculating the fluid makespan for these average processing times, and let T^* now denote the estimated machine lower bound. Without loss of generality, assume the bottleneck is machine 1.

Order the activities of all the job types in a single sequence which is consistent with the order of the individual routes (this can be done for example by concatenating all the steps of routes $1, \ldots, M$). Redefine the job shop as a single type job shop with the single combined route (a re-entrant line) with ordered steps $k = 1, \ldots, K$, and order the $N = \tilde{N}$ modified jobs as $n = 1, \ldots, N$. Processing at each machine will be prioritized by this order of the jobs.

Determine safety stocks for each activity. These should be of order of magnitude $S_k = O(\log N)$.

Perform the schedule in 3 phases.

Buildup phase Build up safety stocks, by processing enough job steps so that machine i will have S_k jobs waiting for the performance of step k for every $k \in \mathcal{K}_i$. This step will require time T_0 (estimated as $O(\log N)$).

Main phase, with cyclic schedule Perform N cycles, in which each machine processes exactly one job from each activity $k \in \mathcal{K}_i$. The timing of the cycles is determined by machine 1, which performs successive cycles without any idling. The remaining machines start each cycle at the completion of the previous cycle on that machine, or when machine 1 starts it next cycle, whichever is later. This main phase ends when machine 1 is forced to idle, because it has no available jobs to perform a full cycle. This phase will require time $\tilde{T}^*$ (estimated as close to T^*).

Runout phase Complete all the remaining jobs in some arbitrary nonidling fashion. This phase will require time T_1 (estimated as $O(\log N)$).

This policy is similar to the Drum Buffer Rope policy of Goldratt and Cox (1987), which is intended for flow shops: here the bottleneck keeps a buffer with safety stock which prevents it from idling. The buffer is fed by the other machines to keep the level of the safety stocks, the bottleneck determines the cycle time, and the other machines idle when they complete their stocking up of the buffer. Dai and Weiss use this idea in a job shop setting.

We make a few more comments on this policy.

The schedule is online and decentralized: In particular, in the main phase each machine picks up the next job by order of jobs, and the only common signal which is needed is the completion time of each cycle at the bottleneck machine, which permits the other machines to start the next cycle.

Jobs are not necessarily performed according to first-in first-out (FIFO) order at each machine. The priority order of jobs is according to their initial numbering from 1 to N. This is called time stamping, and is usually better than an FCFS policy in multi-operation manufacturing environments or in a multi-class queuing network. It has the advantage that jobs can be ordered by due dates.

Within each machine, serving jobs according to time stamp priorities results in a partial last buffer first served (LBFS) policy. This happens because those jobs which are waiting for later activities will have an earlier time stamp than jobs that are waiting for activities which are earlier in the processing route; therefore, they will be served first.

By choosing the size of the safety stocks as $S_k = O(\log N)$, the probability that any machine will exhaust its safety stock is kept under control.

For technical reasons Dai and Weiss make the following assumption about the processing time distributions.

Assumption 4.1 (Existence of exponential moments) *The probability distributions of the random processing times $X_{m,j}(n)$ satisfy*

$$\mathbb{E}\big(e^{\theta X_{m,j}(1)}\big) < \infty \quad \textit{for some } \theta > 0,$$

which ensures that the $\mathbb{P}(X_{m,j}(n) > x)$ decreases exponentially fast.

This assumption is valid in many cases; in particular, it is reasonable for processing times in a manufacturing or service environment. It may however not hold in computing or data communication systems, where file sizes and therefore also transmission and computing times may be extremely variable, and consist of items of many differing scales (referred to as mice and elephants in the computing literature). Dai and Weiss prove the following.

Theorem 4.2 *Consider random instances of the job shop with fixed routes. Assume that there is a single bottleneck machine, and that the processing time distributions possess finite exponential moments. Let N denote the multiplicity. Let T^{Opt} denote the (random) optimal makespan and T^H denote the (random) makespan of the safety stock offset cyclic schedule, with safety stocks*

$$S_{m,j} = \lceil c_2 \log N \rceil \quad \textit{for } j = 2, \ldots, r_m,\ m = 1, \ldots, M.$$

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

$$\mathbb{P}\big\{T^H - T^{\mathrm{Opt}} \geq c_1 \log N\big\} \leq 1/N \quad \textit{for all } N \geq 1.$$

In practical terms this is asymptotically optimal as $N \to \infty$, with an additive optimality gap of $O(\log N)$.

The reason for this $O(\log N)$ additive optimality gap is that the levels of the safety stocks fluctuate over time much like the queue length process in a queuing system. If the safety stocks are exhausted, the cyclic schedule is disrupted. Hence, to avoid the safety stocks from being exhausted, initial safety stocks are set to be larger than the largest fluctuation. This largest fluctuation of the safety stocks, during the scheduling of N cycles, corresponds to a maximal queue length in a queuing system while it serves N customers. The latter is of order $O(\log N)$ if traffic intensity is less than 1, and if processing times possess exponential moments. Hence, under the assumptions of Theorem 4.2 the necessary safety stocks are $O(\log N)$.

5 An on-line tracking rule for general large volume job shops

The large volume job shops of Sect. 4 assumed a fixed set of M routes, with all the N jobs following those same routes

as N becomes large. In this section we generalize the setup to jobs which follow arbitrary routes of bounded length. We present a scheduling rule which does not make any assumptions regarding the structure of the job shop problem and is extremely easy to implement. This schedule was briefly discussed under the name of greedy fluid algorithm (GFA) by Boudoukh et al. (2001). Our schedule is based on the classification of the jobs into classes, $k \in \mathcal{K}$, as described in Sect. 3. Our scheduling rule attempts to track the fluid solution, in a way which is similar to the greedy tracking algorithm of Bertsimas and Sethuraman (2002) described in Sect. 4.3.

We achieve the tracking by assigning on-line priorities to classes of jobs at any time t. Recall that we use $Q_k(t)$ to denote the count of jobs in class k at time t, and we use $Q_k^+(t)$ to denote the count of all the activities of class k which are still to be completed after t. The priority of class k at time t is determined by comparing $Q_k^+(t)$ with $q_k^+(t)$, which is the amount of fluid that still needs to go through class k at time t, as given in the fluid solution (7). Define the lag-fraction for class k at time t as

$$\frac{Q_k^+(t) - q_k^+(t)}{q_k^+(t)}. \tag{8}$$

This measures by how much the actual schedule of jobs of class k is lagging behind the optimal fluid solution, measured as a fraction of the fluid solution value. Classes which have a higher value of lag-fraction have higher priority.

The lag-fraction is equivalent to a simpler priority index, which is easily derived. Recall that $Q^+(0) = q^+(0)$, and use (7) to write

$$\frac{Q_k^+(t)}{Q_k^+(0)} = \left(\frac{Q_k^+(t) - q_k^+(t)}{q_k^+(t)} + 1\right) \Big/ \left(1 - \frac{t}{\mathbf{T}^*}\right),$$

which leads to the following.

An online fluid tracking policy At each time t and for each machine i that is free, denote $\mathcal{K}_i(t) = \{k \in \mathcal{K}_i : Q_k(t) > 0\}$. This is the set of available activities of machine i. If $\mathcal{K}_i(t) = \emptyset$, machine i idles. Otherwise, machine i will process an activity of class k of one of the available jobs, where

$$k \in \arg \max_{k' \in \mathcal{K}_i(t)} \frac{Q_{k'}^+(t)}{Q_{k'}^+(0)}. \tag{9}$$

When the argmax is not unique, choose k using an arbitrary tie-breaking rule. When there are several jobs of class k, choose one using an arbitrary rule.

As we can see, only two things are necessary for this policy: first we need to classify the activities of all the jobs into a finite fixed number of classes, and second we need to obtain $Q_{k'}^+(0)$, the total number of times that activities of each class k will be performed. Note also that we can use equivalently

$$k \in \arg \min_{k' \in \mathcal{K}_i(t)} \frac{S_{k'}^+(t)}{Q_{k'}^+(0)},$$

where we only need to keep track of service completion of activities of each class k'.

If all the jobs follow a fixed set of routes, a natural classification is simply the union of the steps of all the routes. In the 10×10 example this would lead to 100 classes. For each of the $m = 1, \ldots, M$ routes there would then be N_m jobs in the large volume job shop, and all the classes of route m will have $Q_{k'}^+(0) = N_m$. A more general situation is a job shop in which each machine (as is the case in a manufacturing environment) has a fixed set of activities that it does, and each job has some individual fixed sequence of activities, of a bounded length. The number of possible routes could then be much larger than the number of jobs even in a large volume job shop. For example, for the 10×10 model, we could assume that each of the 10 machines has 10 activities which it can perform, and each job consists of 10 steps that go through all the machines. The number of 10 step routes which visit each machine exactly once is: $10!10^{10} \approx 3.6 \times 10^{16}$ routes. In our simulations we experimented with job shops of up to $10 \cdot 2^{16}$ jobs. We note that $10!10^{10}/(10 \cdot 2^{16}) \approx 5^{10}$ so the number of jobs is negligible compared to the number of routes. To obtain $Q_{k'}^+(0)$ in that case we would need to count the visits of all the jobs to each class.

More generally, we can apply our fluid tracking policy also if $Q_{k'}^+(0)$ is not known exactly, by using an estimate. For example, if job routes are stochastic, and we know what proportion of jobs starts at each k, given by a row probability vector $(\alpha_k)_{k=1}^K$, and we know the routing probabilities $P_{k,k'}$ from class k to k', then for a total of N jobs we can estimate $(Q_1^+(0), \ldots, Q_K^+(0)) = N\alpha(I - P)^{-1}$.

The fluid tracking policy is more general than the policies which we surveyed in Sect. 4. It is interesting to note that when it is applied to the job shops that were considered in Sect. 4 it performs similarly to them. We go into more details now.

Proposition 5.1

(i) *If the job shop is a re-entrant line in which all the jobs follow a single route* (*which may revisit any machine several times*), *then the fluid tracking policy will be equivalent to LBFS, and will give highest priority to jobs that are nearest to completion.*

(ii) *If the job shop has M fixed routes, then jobs on each route will be served according to LBFS at each machine.*

Proof If jobs from a single route are waiting as class k and as class k' at machine i, and k (k') is step j (j') along the

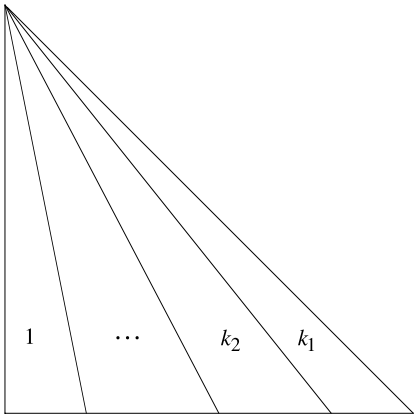

Fig. 4 The form of the fluid limit of the fluid tracking policy in a re-entrant line

route, with $j < j'$ (k' is downstream, further along the route than k) then of necessity $Q_k^+(t) < Q_{k'}^+(t)$, so k' always has priority over k. □

Similar to the greedy tracking algorithm of Bertsimas and Sethuraman (2002) of Sect. 4.3, our rule may schedule jobs earlier than in the fluid solution. To illustrate this, consider processing of jobs in a re-entrant line job shop according to our general greedy tracking algorithm. The schedule which is obtained does not in fact resemble the fluid picture of Fig. 3. Rather, it follows in the fluid limit the picture described in Fig. 4, as we now explain.

For a re-entrant line define the following sequence of successive bottleneck classes: let i be the bottleneck machine, in the sense that $\sum_{k\in\mathcal{K}_i} m_k Q_k^+(0)$ is maximal among all the machines. Denote as k_1 the first class served by machine i. Define $k_1 > k_2 > \cdots > k_L = 1$ inductively: following k_l consider only the work in classes $1,\ldots,K_i$, locate the bottleneck machine for this work only, and let k_{l+1} be the first buffer of that machine. Work will accumulate in these bottleneck classes, which will be working essentially without idling, until there is no more work for them in the job shop.

Consider now a job shop with fixed routes and random processing times as discussed by Dai and Weiss (2002). Comparison of our fluid tracking policy with the safety stock offset cyclic schedule of Dai and Weiss shows the following: for a job shop with fixed stochastic routes and total of N jobs, under the assumptions of Theorem 4.2, the gap between the fluid lower bound and the heuristic value is with high probability $O(\log N)$.

6 The job shop scheduling software package

In order to evaluate the policy proposed in the previous section, we have implemented a software package that allows for simulation of large volume job shop problems. This package is called the Job Shop Simulation Package (JSSP). It is available at http://stat.haifa.ac.il/~yonin/thesis/jobshopsim/shopsim.html. The JSSP simulates job shop realizations with a fixed number of routes having arbitrary multiplicity. It implements a discrete event simulation. Processing times of jobs are either deterministic or may be generated as random variables. The scheduling rules include the fluid tracking policy that we describe here as well as several other rules such as buffer priority policies. An input file specifies the structure of the job shop problem including mean processing times for each operation.

The JSSP is deployed in several configurations. First, it may be run in batch mode. This allows for mass simulation of job shop realizations. We have used this mode to collect some of the simulation results which we present below. Second, the JSSP may be run using an interactive graphic user interface. In this mode, it is possible to visually inspect the evolution of the job shop. The visual display includes a toy graphical model of the job shop and a Gantt chart which is continuously updated as the simulation progresses. See Fig. 5. There is no limitation to use a single model, any job shop problem may be displayed by properly defining an input file. The third mode is somewhat similar, but operates in a different setting: in a web browser as a JAVA applet. All of the above configurations make this package suitable for both demonstration and empirical research.

7 Simulation experiments with the on-line tracking rule

We have used the JSSP to evaluate the performance of large job shop scheduling rules. We have run a total of approximately 50,000 simulation runs of various configurations. The total execution time of all simulations was more than two months (on a PC). And the collection and ordering of the data was quite involved. The simulation study is fully described in Nazarathy (2001).

Our primary purpose was to evaluate the effectiveness of the on-line tracking rule of Sect. 5 on a massive scale. As we saw in Sect. 4, the performance of job shop schedules based on fluid heuristics can be expected to be asymptotically optimal when the volume of jobs is large. If the job shop consists of a fixed number of job types that have identical routes and processing times, then the cyclic heuristic achieves a makespan that is within a constant bound from the fluid lower bound, for any volume of jobs. Dai and Weiss (2002) show that for a fixed set of routes and jobs with randomly generated processing times, under the assumption of a single bottleneck machine, and existence of exponential moments, the gap between a fluid heuristic and lower bound can be bounded by a $O(\log N)$ for a volume of N jobs. We expect our tracking rule to perform similarly. In fact, we found that it performed better than the $O(\log N)$ guaranteed by Dai and Weiss: even when relaxing the assumption

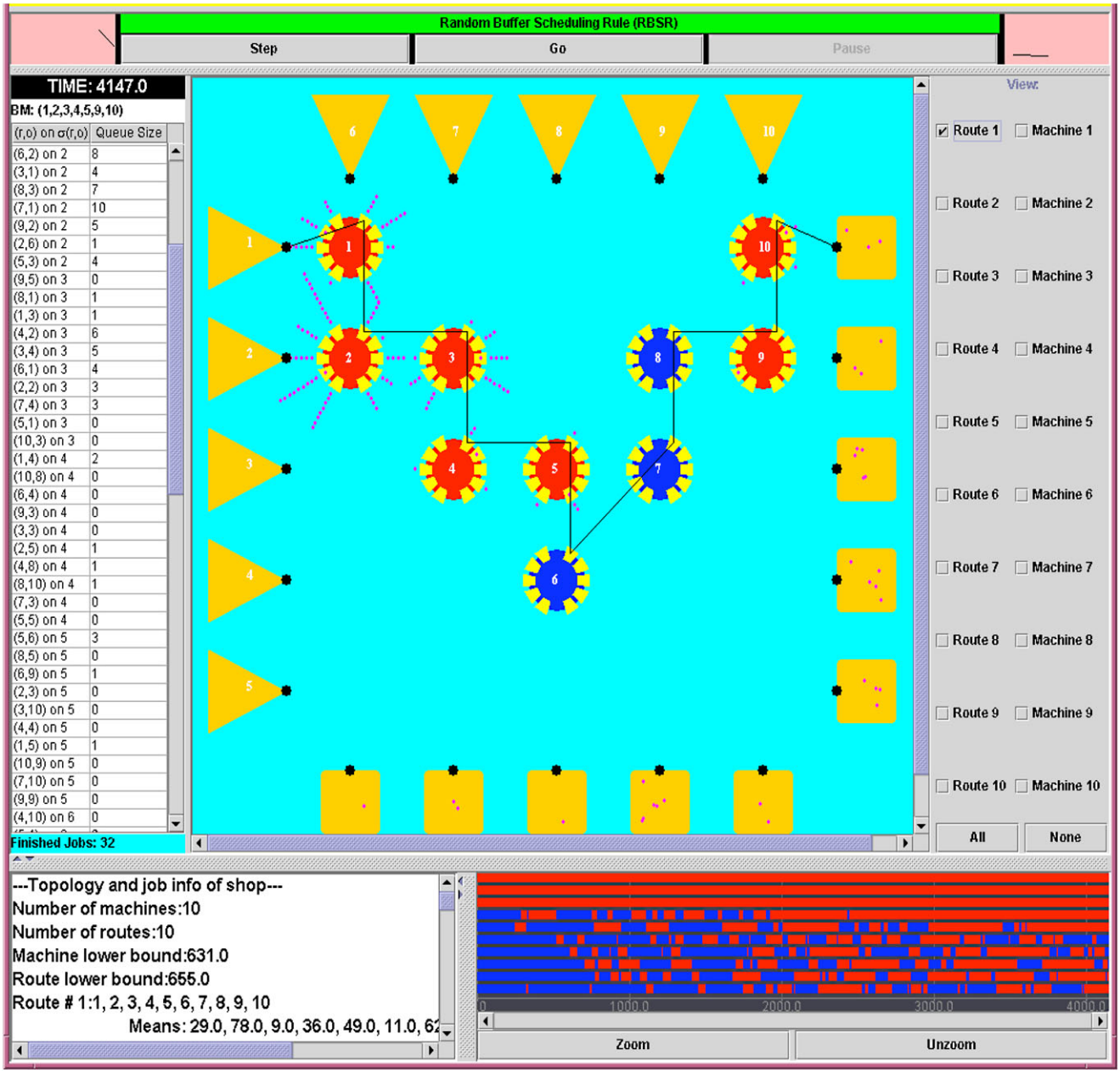

Fig. 5 (Color online) A screen shot of the JSSP. This picture represents a simulation of the 10 by 10 problem. The simulation state is summarized in the left-hand side of the screen and is graphically displayed in the center of the screen. Each *circle* represents a machine. *Red circles* are in operation and blue are idling. *Yellow widgets on the circles* represent buffers of jobs. *Triangles* represent initial jobs and *squares* represent finished jobs. The *tiny red dots* are actual jobs. The Gantt chart at the bottom right is continuously updated and visually displays the utilization over time of each of the machines

of existence of exponential moments, we found that the gap between the heuristic and the fluid lower bound behaves like a constant which does not grow with the volume of jobs.

The instances of large volume job shops which we have simulated are mostly based on the 10×10 problem of Muth and Thompson. In addition, we have made use of a 15×20 problem that was used in Adams et al. (1988) (see Table 1, Problem 9, of that paper) and a 10×50 problem used in Storer et al. (1992) (see Table 3, Problem "10×50 Hard", of that paper). In all 3 problems each of the jobs need to pass through all machines. We shall refer to these three problems as MT, ABZ, and SWV respectively. Our method of generating job shop problem instances was to take the original problem (MT, ABZ, or SWV) and generate many jobs that follow the original routes. For example, for the SWV problem, in each experiment the basic unit consists of 50 jobs,

each with 10 activities, on all 10 machines. For each experiment we used several multiplicities, N, with the values $N = 2^l$ where $l = 1, \ldots, 15$ (some of the experiments were only run for smaller l). We used stochastic randomly generated processing times in the experiments, and so we have used between 15 and 50 replicates for each multiplicity of each experiment.

In each simulation replicate we used our fluid tracking policy of Sect. 5 to schedule the jobs. We used the following tie-breaking rules: we ordered the classes $k = 1, \ldots, K$. Whenever two classes at a machine had the same priority value, we took the class with the lower k. We ordered the jobs in each class as follows: jobs which were in the class initially (at time 0) were ordered in line arbitrarily. Any jobs arriving later into the class joined the end of the line. We then used head-of-the-line priority within each class and simulated the following 3 quantities:

- The fluid solution machine lower bound $\mathbf{T}^*$. This is the total busy time of the bottleneck; it is random for stochastic processing times.
- The completion time of the last job on the bottleneck machine, $\mathbf{T}^c$. At this time there is no more work for the bottleneck machine anywhere in the job shop.
- The makespan of the schedule, $\mathbf{T}^H$.

By the design of our fluid tracking heuristic we expected the bottleneck machine to be busy most of the time. However, at the start of the schedule there is a period in which it may be starved of work, until safety stocks are built up, cycles appear, and pipelining occurs. This initial idling of the bottleneck is included in the difference $\mathbf{T}^c - \mathbf{T}^*$. There follows a long period in which the bottleneck machine is busy with no interruptions, until almost the time that it completes the processing of the last job. The remainder of the schedule is the time that is required of all the other machines to complete all the processing, and is given by $\mathbf{T}^H - \mathbf{T}^c$. We refer to the first of these as the *starve time* of the bottleneck and to the second as the *runout time*. We refer to their sum $\mathbf{T}^H - \mathbf{T}^*$ as the *gap*. As discussed in Sect. 3, it is an upper bound on the suboptimality of our heuristic.

Note that the expected fluid makespan increases linearly with the multiplicity N. For the original 10×10 problem this is $631N$, and is equal to $\approx 10^7$ for $N = 2^{14}$. Mostly we obtained gaps of no more than 4000 (and often much less). Thus the suboptimality of our scheduling rule for large volume jobs shops is negligible!

We now describe our experiments.

7.1 Experiments on sensitivity to the distribution of processing times

We took instances of the MT10 and used the processing times (see Table 1) as means, and generated the activity processing times of each job out of some given distributions with these means. The different distributions which we used and the arguments for experimenting with each of them were as follows.

Deterministic: All jobs of the 10 types are identical with the original 10×10 data. The gap converges to a constant value of 447, for all $N \geq 10$. Of course, there is no need for replicates.
Gaussian, $CV = 0.25$: We use Gaussian distribution for all activities, with standard deviation $\sigma = 0.25\mu$, i.e., coefficient of variation of one quarter. This is a reasonable and realistic distribution for processing times (we truncated at 0 in the rare events of negative values). Typical gaps converged to ∼500.
Exponential: This has $CV = 1$ and is rather more variable than usual for manufacturing. Typical gaps converged to ∼1300.
Weibull with shape parameter $\frac{1}{2}$: This distribution is even more variable ($CV = 2.2$). Furthermore, while Weibull distributions with shape parameter <1 possess finite moments of every order, they do not possess finite exponential moments, since their density decreases more slowly than any exponential. This means that the results of Dai and Weiss (2002) do not apply.
Pareto 3: This distribution possesses finite mean and variance, but all higher moments are infinite. Again the results of Dai and Weiss (2002) do not apply.
Pareto 2: This distribution possesses finite mean but infinite variance (and hence infinite higher moments). Results of Dai and Weiss (2002) do not apply.

The results of this experiment are summarized in Fig. 6: there are 6 figures for the six different distributions, which show the values of the gap. The horizontal axes in each figure give the multiplicity on a logarithmic scale, for multiplicities of $N = 2^l$. At each multiplicity we plot the gap for each replicate as a dot, and the heavy green line with dots gives the average gap, over the replicates. The range between the light green lines above and below the average are the 5% and 95% empirical values for the replicates. We also plotted in the heavy blue line the average starve time. The runout time is then given by the height between the gap and starve time.

Our conclusions from this experiment are:

- The gap for all 6 types of distributions converges to a constant, hence is seems that the suboptimality is bounded by a constant for all 6 distributions.
- It seems that the bound of $O(\log N)$ for the gap, shown by Dai and Weiss (2002), is conservative, and the requirement of exponential moments is not needed.
- The runout time is significantly smaller than the starve time, and converges faster with N.

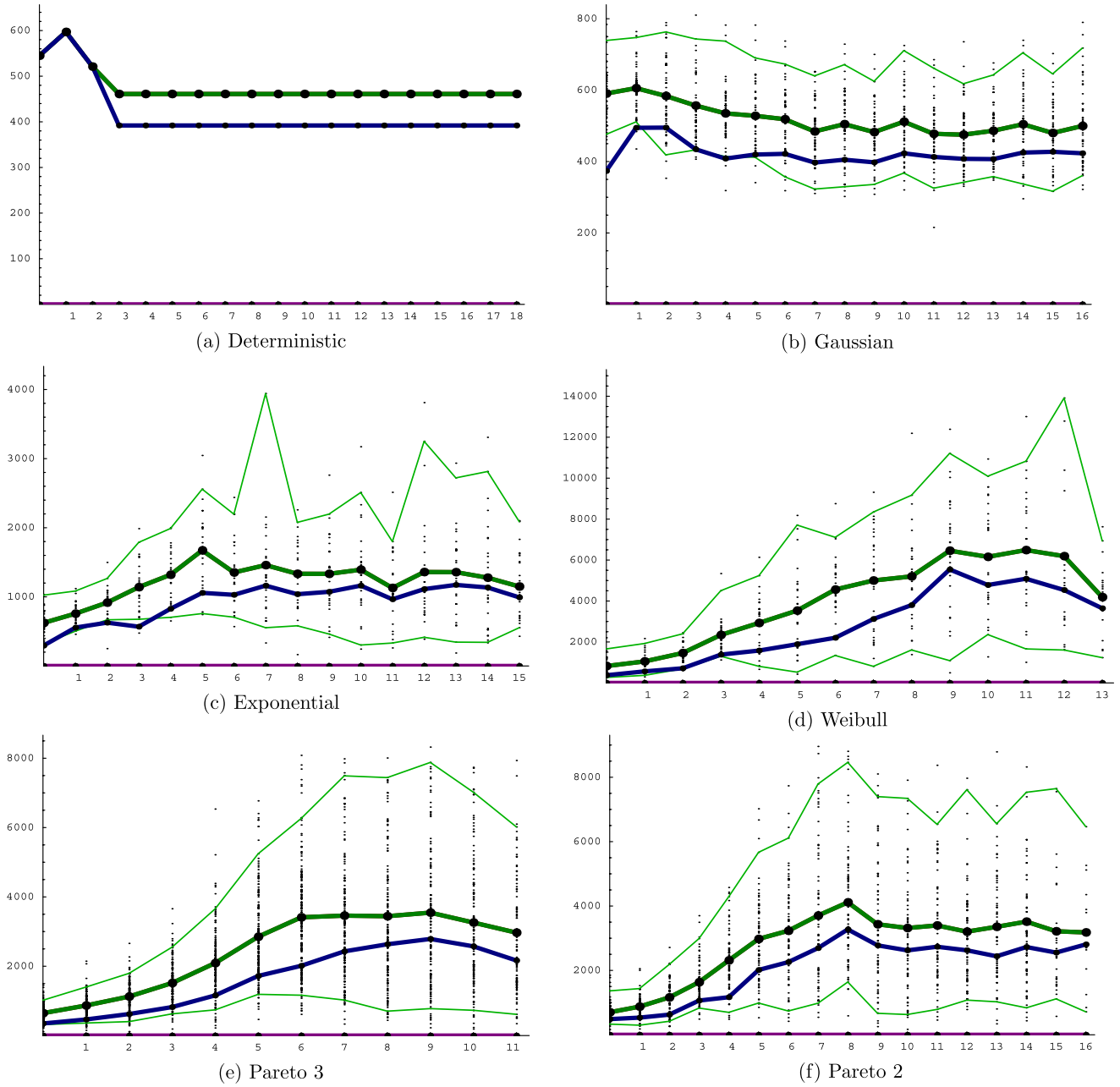

Fig. 6 Gap as a function of $\log_2(N)$ as estimated by simulation for MT high-multiplicity problem with varying processing time distributions

- Observation of the actual runs confirms that the bottleneck machine is busy continuously through most of the schedule, after an initial period which almost equals the starve time, and until the bottleneck machine finishes all its work.

7.2 Experiments on alternative job shop structures

We also performed similar experiments to the ones described above, on large volume job shops that are generated from SWV and ABZ. Our findings were consistent with those of MT10: The suboptimality of our fluid heuristic is bounded by a constant, irrespective of the distribution used. For these cases we have simulated deterministic, exponential, and Pareto 2 problem instances similar to what is described above. Our results are given in Fig. 7.

Similarly to the more detailed MT results, the suboptimality gap again appears to be a constant.

7.3 Experiments with multiple bottlenecks

In the 10×10 MT example, machine 4 is the unique bottleneck machine, with fluid makespan of 631. A single bottle-

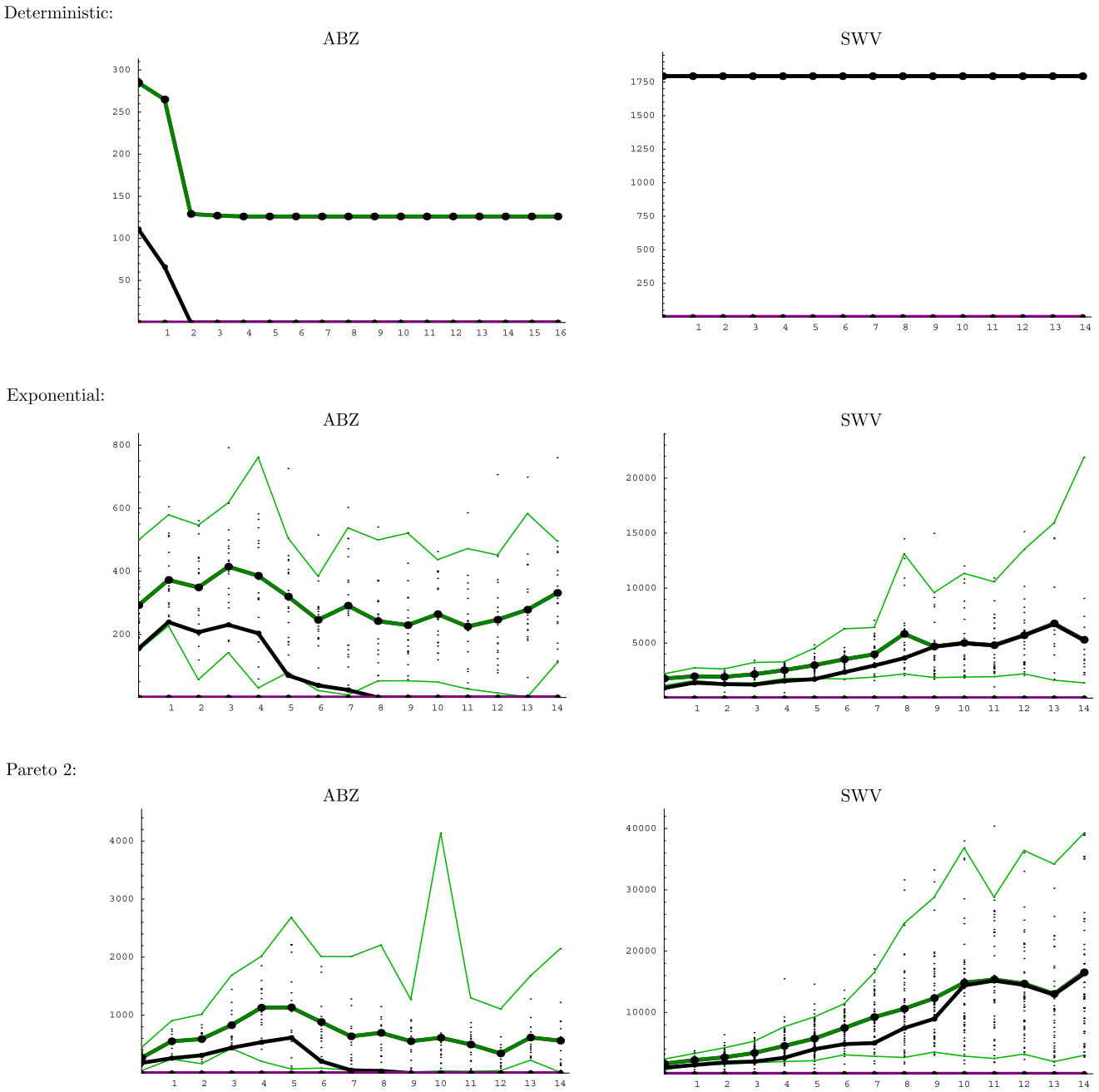

Fig. 7 Gap as a function of $\log_2(N)$ as estimated by simulation for the ABZ and SWV high-multiplicity problem with varying processing time distributions

neck has been one of the requirements for Theorem 4.2 of Dai and Weiss (2002). To test the effect of multiple bottlenecks we have modified the 10×10 data to obtain a problem in which all the machines were bottlenecks, by stretching all the processing times of the nonbottleneck machine activities. The new values of the 100 processing times were then used as new means for generating simulation runs.

We performed two experiments. In the first we left all the routes as in the original 10×10 problem. In this experiment all 10 machines were bottlenecks but all the jobs were initially, at time 0, located in classes of 3 of the machines, machines 1, 2, and 3 (see Table 1). In the second experiment we did a cyclic rearrangement of the steps of each route so that the job on route $m = 1, \ldots, 10$ started with an activity at machine m.

In the first experiment we found that the gap increased with the multiplicity N, faster than $O(\log N)$ but slower than $O(N)$. Figure 8 summarizes this experiment: it shows the ratio of the gap to N, on a logarithmic scale of N. It seems that this does converge to 0 as N increases, hence the

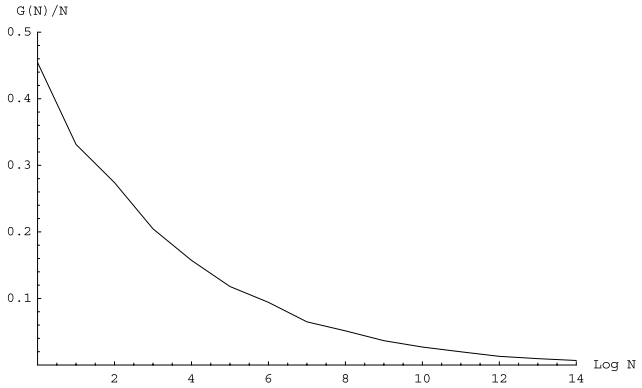

Fig. 8 Balanced job shop (all machines are bottleneck). The mean relative gap on a logarithmic scale of N. Processing times are taken from an exponential distribution

heuristic is still asymptotically optimal. We repeated this experiment for the various distributions, with similar results. In Fig. 8 we used exponential processing times. When a queuing network is working at close to full utilization, and there is more than one bottleneck, the system is said to be in balanced heavy traffic. We conclude that our heuristic performs less well in a situation of balanced heavy traffic.

In the second experiment, when all 10 machines were bottlenecks but each of the machines starts with N initial jobs, the behavior was entirely different: the gap seemed to converge to a constant. Under these conditions the job shop behaves like a queuing network with infinite virtual queues, as discussed by Weiss et al. (2008), Nazarathy and Weiss (2008, 2009), Weiss (2005). We conclude that with an ample supply of initial work the heuristic is effective also when there are multiple bottlenecks.

8 Concluding remarks

We have surveyed some previous work in which it was shown that for a large volume job shop it is possible to achieve a makespan which is very close to the machine lower bound. We have posed the fluid job shop minimum makespan problem and solved it under some general assumptions, and found that the optimal fluid makespan equals the machine lower bound $\mathbf{T}^*$, and can be achieved by decreasing all the fluid volumes in the job shop linearly, from their initial state at time 0, to 0 at time $\mathbf{T}^*$.

We then introduced our fluid tracking policy, which gives priority to jobs of a class with the highest $Q_k^+(t)/Q_k^+(0)$. This policy is very simple to implement, it requires very little prior information, and it is decentralized and on-line.

We note nevertheless that the minimum makespan problem is somewhat degenerate. Clearly, the fluid problem has many optimal solutions, and with a large volume job shop there are very many schedules, which are very different, and yet achieve very close to the optimal makespan.

The idea of tracking a fluid solution can however also be used for other problems. In Nazarathy and Weiss (2009), Weiss (1999) we use it to minimize weighted flowtime in a multi-class queuing network over a finite time horizon. This can be implemented as a heuristic for minimum weighted flowtime in a job shop.

We believe that our approach of solving a fluid problem and tracking the fluid solution by an online decentralized policy provides a bridge between deterministic scheduling theory and the control of steady state queuing networks. We hope that it will prove useful in practice, in large volume job shops.

Acknowledgement We are grateful to an anonymous referee for many helpful comments.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, *34*, 391–401.

Balas, E. (1968). Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations Research*, *17*, 941–957.

Barany, I. (1981). A vector sum theorem and its application to improving flow shop guarantees. *Mathematics of Operations Research*, *6*, 445–452.

Bertsimas, D., & Gamarnik, D. (1999). Asymptotically optimal algorithms for job shop scheduling and packet routing. *Journal of Algorithms*, *33*, 296–318.

Bertsimas, D., & Sethuraman, J. (2002). From fluid relaxations to practical algorithms for job shop scheduling: the makespan objective. *Mathematical Programming*, *1*, 61–102.

Bertsimas, D., Gamarnik, D., & Sethuraman, J. (2003). From fluid relaxations to practical algorithms for job shop scheduling: the holding cost objective. *Operations Research*, *51*(5), 798–813.

Boudoukh, T., Penn, M., & Weiss, G. (1998). Job-shop—an application of fluid approximation. In I. Gilad (Ed.), *Proceedings of the tenth conference of industrial engineering and management*, pp. 254–258. June 1998, Haifa Israel.

Boudoukh, T., Penn, M., & Weiss, G. (2001). Scheduling job shops with some identical or similar jobs. *Journal of Scheduling*, *4*, 177–199.

Carlier, J., & Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, *35*(2), 164–176.

Chen, H., & Yao, D. D. (2003). *Fundamentals of queuing networks, performance, asymptotics and optimization*. New York: Springer.

Chen, M., Pandit, C., & Meyn, S. (2003). In search of sensitivity in network optimization. *Queuing Systems Theory and Applications*, *44*, 313–363.

Conway, R. W., Maxwell, W. L., & Miller, L. W. (1967). *Theory of scheduling*. Reading: Addison-Wesley.

Dai, J. G., & Lin, W. (2005). Maximum pressure policies in stochastic processing networks. *Operations Research*, *53*, 197–218.

Dai, J. G., & Lin, W. (2006, submitted). Asymptotic optimality of maximum pressure policies in stochastic processing networks. *Annals of Applied Probability*.

Dai, J. B., & Weiss, G. (2002). A fluid heuristic for minimizing makespan in job-shops. *Operations Research*, *50*, 692–707.

Demers, A., Keshav, S., & Shenker, S. (1990). Analysis and simulation of a fair queuing algorithm. *Internetworking Research and Experience, 1*.

Gantt, H. L. (1910) *Work, wages and profit*. The Engineering Magazine, New York, 1910; republished as *Work, wages and profits*, Easton, Pennsylvania, Hive Publishing Company, 1974, ISBN 879600489.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: a guide to the theory of NP-completeness*. San Francisco: Freeman.

Gittins, J. C. (1979). Bandit processes and dynamic allocation indices. *J Royal Statistical Society Series B*, *14*, 148–167.

Goldratt, E. M., & Cox, J. (1987). *The goal: excellence in manufacturing*. Croton-on-Hudson: North River Press.

Greenberg, A. G., & Madras, N. (1992). How fair is fair queuing? *Journal of the Association for Computing Machinery*, *39*, 568–598.

Hanen, C. (1994). Study of a NP-hard cyclic scheduling problem: the recurrent job-shop. *European Journal of Operational Research*, *72*, 82–101.

Harrison, J. M. (1988). Brownian models of queuing networks with heterogeneous customer populations. In W. Fleming, & P. L. Lions (Eds.), *Proceedings of the IMA workshop on stochastic differential systems*. New York: Springer.

Harrison, J. M., & Van Mieghem, J. (1997). Dynamic control of Brownian networks: state space collapse and equivalent workload formulations. *Annals of Applied Probability*, *7*, 747–771.

Henderson, S. G., Meyn, S. P., & Tadic, V. B. (2003). Performance evaluation and policy selection in multiclass networks. *Discrete Event Dynamic Systems*, *13*(1–2), 149–189.

Hochbaum, D. S., & Shamir, R. (1991). Strongly polynomial algorithms for high multiplicity scheduling problem. *Operations Research*, *39*, 648–653.

Jansen, K., Solis-Oba, R., & Srividenko, M. (2000). *Makespan minimization in job shops, a linear time approximation scheme*. Preprint, see also preliminary versions in STOC'99 and APPROX'99.

Klimov, G. P. (1974). Time sharing service systems I. *Theory of Probability and Applications*, *19*, 532–551.

Kopzon, A., Nazarathy, Y., & Weiss, G. (2008). *A push pull system with infinite supply of work*. Preprint.

Kulak, O., Yilmaz, I. O., & Gŭnther, H. O. (2007). PCB assembly scheduling for collect-and-place machines using genetic algorithms. *International Journal of Production Research*, *45*(17), 3949–3969.

Kushner, H. J. (2001). *Heavy traffic analysis of controlled queuing and communication networks*. Berlin: Springer.

Kushner, H. J., & Martins, L. F. (1990). Routing and singular control for queuing networks in heavy traffic. *SIAM Journal on Control and Optimization*, *28*, 1209–1233.

Kushner, H. J., & Ramachandran, K. M. (1989). Optimal and approximately optimal control policies for queues in heavy traffic. *SIAM Journal on Control and Optimization*, *27*, 1293–1318.

Lawler, E. L., Lenstra, J. K., Rinnoy Kan, A. H. G., & Shmoys, D. B. (1993). In S. C. Graves, A. H. G. Rinnoy Kan, & P. H. Zipkin (Eds.), *Sequencing and scheduling: algorithms and complexity, in logistics of production and inventory*. New York: Elsevier Science.

Martin, P., & Shmoys, D. B. (1996). A new approach to computing optimal schedules for the job-shop scheduling problem. In *International IPCO conference* (pp. 389–403).

Matsuo, H. (1990). Cyclic sequencing problems in the two-machine permutation flow shop: complexity, worst-case and average-case analysis. *Naval Research Logistics*, *37*, 679–694.

McCormick, S. T., Pinedo, M. L., Shenker, S., & Wolf, B. (1989). Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, *37*, 925–935.

Meyn, S. P. (2001). Sequencing and routing in multiclass queuing networks, part I: feedback regulation. *SIAM Journal on Control and Optimization*, *40*, 741–776. IEEE International Symposium on Information Theory, Sorrento, Italy, June 25–June 30, 2000.

Meyn, S. P. (2008). *Control techniques for complex networks*. Cambridge: Cambridge University Press.

Moallemi, C. C., Kumar, S., & Van Roy, B. (2008). *Approximate and data-driven dynamic programming for queuing networks*.

Muth, J. F., & Thompson, G. L. (1954). *Industrial scheduling*. Prentice-Hall: Englewood Cliffs.

Nazarathy, Y. (2001). *Evaluation of on line scheduling rules for high volume job shop problems: a simulation study*. MA thesis, The University of Haifa, Israel.

Nazarathy, Y., & Weiss, G. (2008). *Positive Harris recurrence and diffusion scale analysis of a push pull queuing network*. Preprint.

Nazarathy, Y., & Weiss, G. (2009). Near optimal control of queuing networks over a finite time horizon. *Annals of Operations Research*, *170*, 233–249.

Ni-o-Mora, J. (2001). Restless bandits, partial conservation laws and indexability. *Advances in Applied Probability*, *33*, 76–98.

Ni-o-Mora, J. (2002). Dynamic allocation indices for restless projects and queuing admission control: a polyhedral approach. *Mathematical Programming, Series A*, *93*, 361–413.

Ni-o-Mora, J. (2006). Restless bandit marginal productivity indices, diminishing returns and optimal control of make-to-order/make-to-stock M/G/1 queues. *Mathematics of Operations Research*, *31*, 50–84.

Papadimitriou, C. H., & Tsitsiklis, J. N. (1999). The complexity of optimal queuing network control. *Mathematics of Operations Research*, *24*(2), 293–305.

Parekh, A. K., & Gallager, R. G. (1993). A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, *1*, 344–357.

Parekh, A. K., & Gallager, R. G. (1994). A generalized processor sharing approach to flow control in integrated services networks: the multiple-node case. *IEEE/ACM Transactions on Networking*, *2*, 137–150.

Pinedo, M. (2002). *Scheduling, theory, algorithms and systems* (3rd edn.). New York: Springer.

Roundy, R. (1992). Cyclic scheduling for job shops with identical jobs. *Mathematics of Operations Research*, *17*, 842–865.

Sevastyanov, S. V. (1987). Bounding algorithm for the routing problem with arbitrary paths and alternative servers. *Cybernetics*, *22*, 773–781.

Sevastyanov, S. V. (1994). On some geometric methods in scheduling theory, a survey. *Discrete Applied Mathematics*, *55*, 59–82.

Sevastyanov, S. V., & Woeginger, G. J. (1998). Makespan minimization in open shops, a polynomial type approximation scheme. *Mathematical Programming*, *82*, 191–198.

Shmoys, D. B., Stein, C., & Wein, J. (1994). Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, *23*, 617–632.

Stallings, W. (2007). *Data and computer communications* (8th edn.). Upper Saddle River: Prentice-Hall.

Stolyar, A. L. (2004). MaxWeight scheduling in a generalized switch: state space collapse and equivalent workload minimization under complete resource pooling. *Annals of Probability*, *14*, 1–53.

Storer, R. H., Wu, S. D., & Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, *38*, 1495–1509.

Tassiulas, L. (1995). Adaptive back-pressure congestion control based on local information. *IEEE Transactions on Automatic Control*, *40*, 236–250.

Van Zant, P. (2004). *Microchip fabrication* (5th edn.). New York: McGraw-Hill.

Veatch, M. H. (2005). *Approximate dynamic programming for networks: fluid models and constraint reduction*. Preprint.

Wein, L. M. (1992). Scheduling networks of queues: heavy traffic analysis of a multistation network with controllable inputs. *Operations Research*, *40*, S312–S334.

Weiss, G. (1995). On the optimal draining of a fluid re-entrant line. In F. P. Kelly, & R. Williams (Eds.), *IMA volumes in mathematics and its applications: Vol. 71. Stochastic networks, proceedings of ima workshop*, Minnesota, February 1994 (pp. 91–104). New York: Springer.

Weiss, G. (1999). Scheduling and control of manufacturing systems—a fluid approach. In *Proceedings of the 37th allerton conference*, (pp. 577–586), Monticello, IL, 21–24 September 1999.

Weiss, G. (2005). Jackson networks with unlimited supply of work and full utilization. *Journal of Applied Probability*, *42*, 879–882.

Whittle, P. (1981). Restless bandits: activity allocation in a changing world. *Journal of Applied Probability 25A*, 287–298.

Williamson, D. P., Hall, L. A., Hoogeveen, J. A., Hurkens, C. A. J., Lenstra, J. K., Sevast'ynaov, S. V., & Shmoys, D. B. (1997). Short shop schedules. *Operations Research*, *45*, 288–294.

Zhang, H. (1995). Service disciplines for guaranteed performance service in packet-switching networks. In *Proceedings of the IEEE, October 1995*.