BETTER APPROXIMATION GUARANTEES FOR JOB-SHOP SCHEDULING*

LESLIE ANN GOLDBERG †, MIKE PATERSON ‡, ARAVIND SRINIVASAN §, AND ELIZABETH SWEEDYK ¶

Abstract. Job-shop scheduling is a classical NP-hard problem. Shmoys, Stein & Wein presented the first polynomial-time approximation algorithm for this problem that has a good (polylogarithmic) approximation guarantee. We improve the approximation guarantee of their work, and present further improvements for some important NP-hard special cases of this problem (e.g., in the *preemptive* case where machines can suspend work on operations and later resume). We also present NC algorithms with improved approximation guarantees for some NP-hard special cases.

Key words. Approximation, guarantees, job-shop, scheduling.

AMS subject classifications. 68Q25, 68R05.

1. Introduction. Job-shop scheduling is a classical NP-hard minimization problem [10]. We improve the approximation guarantees for this problem and for some of its important special cases, both in the sequential and parallel algorithmic domains; the improvements are over the current-best algorithms of Leighton, Maggs & Rao [11] and Shmoys, Stein & Wein [21]. In job-shop scheduling, we have n jobs and m machines. A job consists of a sequence of operations, each of which is to be processed on a specific machine for a specified integral amount of time; a job can have more than one operation on a given machine. The operations of a job must be processed in the given sequence, and a machine can process at most one operation at any given time. The problem is to schedule the jobs so that the *makespan*, the time when all jobs have been completed, is minimized. An important special case of this problem is

*A preliminary version of this work appears in *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 599–608, 1997.

† Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK. `leslie@dcs.warwick.ac.uk`. Part of this work was performed at Sandia National Laboratories and was supported by the U.S. Department of Energy under contract DE-AC04-76AL85000. Part of this work was supported by ESPRIT Projects ALCOM–IT(20244) and RAND-II(21726), by EU Fifth Framework Project IST-1999-14186 — ALCOM–FT, and by EPSRC grant GR/L60982.

‡ Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK. `msp@dcs.warwick.ac.uk`. Part of this work was supported by Projects ALCOM–IT and ALCOM–FT (as above).

§ Bell Laboratories, Lucent Technologies, 600-700 Mountain Avenue, Murray Hill, NJ 07974-0636, USA. `srin@research.bell-labs.com`. This work was done while at: (i) the National University of Singapore, supported in part by National University of Singapore Research Grants RP950662 and RP960620; (ii) Cornell University, Ithaca, NY, USA, supported by an IBM Graduate Fellowship; (iii) the School of Mathematics, Institute for Advanced Study, Princeton, NJ, USA, supported by grant 93-6-6 of the Alfred P. Sloan Foundation to the Institute for Advanced Study; (iv) DIMACS (NSF Center for Discrete Mathematics and Theoretical Computer Science), supported by NSF grant NSF-STC91-19999 to DIMACS and by support to DIMACS from the New Jersey Commission on Science and Technology; (v) the Sandia National Laboratories, New Mexico, USA; (vi) the Department of Computer Science, University of Warwick, Coventry, UK; and (vii) the Department of Computer Science, University of Melbourne, Victoria, Australia, sponsored by a "Travel Grants for Young Asian Scholars" scheme of the University of Melbourne; this part of the work was done while on study leave.

¶ Dept. of Computer Science, Harvey Mudd College, Olin Science Center, 301 E. Twelfth Street Claremont, CA 91711-5980, USA. `z@cs.hmc.edu`. Supported by NSF Research Training Grant. Part of this was done while visiting Sandia National Laboratories.

preemptive scheduling, wherein machines can suspend work on operations, switch to other operations, and later resume the suspended operations (if this is not allowed, we have the *non-preemptive* scenario, which we take as the default); in such a case, all operation lengths may be taken to be one. Even this special case with $n = m = 3$ is NP-hard, as long as the input is encoded concisely [16, 22]. We present further improved approximation factors for preemptive scheduling and related special cases of job-shop scheduling.

Formally, a job-shop scheduling instance consists of jobs $J_1, J_2, \ldots, J_n$, machines $M_1, M_2, \ldots, M_m$, and for each J_j, a sequence of μ_j *operations* $(M_{j,1}, t_{j,1})$, $(M_{j,2}, t_{j,2})$, $\ldots$, $(M_{j,\mu_j}, t_{j,\mu_j})$. Each operation is a (machine, processing time) pair: each $M_{j,k}$ represents some machine M_i, and the pair $(M_{j,i}, t_{j,i})$ signifies that the corresponding operation of job J_j must be processed on machine $M_{j,i}$ for an *uninterrupted* integral amount of time $t_{j,i}$. No machine can process more than one operation at a time; the operations of each given job must be scheduled in the given order. (For each job J_j, the waiting time from the completion of an operation $(M_{j,i}, t_{j,i})$ until the scheduling of $(M_{j,i+1}, t_{j,i+1})$ is allowed to be any non-negative amount.) The problem that we focus on throughout is to come up with a schedule that has a small makespan, for general job-shop scheduling and for some of its important special cases.

1.1. Earlier work. As described earlier, even very restricted special cases of job-shop scheduling are NP-hard. Furthermore, the problem seems quite intractable in practice, even for relatively small instances. Call a job-shop instance *acyclic* if no job has more than one operation that needs to run on any given machine. A single instance of acyclic job-shop scheduling consisting of 10 jobs, 10 machines and 100 operations resisted attempts at exact solution for 22 years, until its resolution by Carlier & Pinson [6]. More such exact solutions for certain instances (with no more than 20 jobs or machines) were computationally provided by Applegate & Cook, who also left open the exact solution of certain acyclic problems, e.g., some with 15 jobs, 15 machines, and 225 operations [3]. The reader is referred to Martin & Shmoys for a recent approach to computing optimal schedules for such problems [14].

Thus, efficient exact solution of all instances with, say, 30 jobs, 30 machines, and 900 operations seems quite beyond our reach at this point; an obvious next question is to look at efficient approximability. Define a ρ-approximation algorithm as a polynomial-time algorithm that always outputs a feasible schedule with a makespan of at most ρ times optimal; ρ is called the approximation guarantee. A negative result is known: if there is a ρ-approximation algorithm for job-shop scheduling with $\rho < 5/4$, then P = NP [23].

There are two simple lower bounds on the makespan of any feasible schedule: $P_{\max}$, the maximum total processing time needed for any job, and $\Pi_{\max}$, the maximum total amount of time for which any machine has to process operations. Recall the definition of acyclic job-shop scheduling given at the beginning of this subsection. For the NP-hard special case of acyclic job-shop scheduling wherein all operations have unit length, a breakthrough was achieved by Leighton, Maggs and Rao in [11], showing that a schedule of makespan $O(P_{\max} + \Pi_{\max})$ always exists! (See Sections 6.1 and 6.2 of Scheideler [17] for a shorter proof of this result.) Such a schedule can also be computed in polynomial time [12]. Feige & Scheideler have presented many new advances in acyclic job-shop scheduling [8].

What about upper bounds for general job-shop scheduling? It is not hard to see that a simple greedy algorithm, which always schedules available operations on machines, delivers a schedule of makespan at most $P_{\max}\Pi_{\max}$; one would however like to aim for much better. Let $\mu = \max_j \ \mu_j$ denote the maximum number of operations per job, and let $p_{\max}$ be the maximum processing time of any operation. By invoking ideas from [11, 19, 20] and by introducing some new techniques, good approximation algorithms were developed in [21]. Their deterministic approximation bounds were slightly improved in [18] to yield the following proposition. (To avoid problems with small positive numbers, henceforth let $\log x$ denote $\log_2 x$ if $x \geq 2$ and 1 if $x < 2$; similarly, let $\log\log x$ denote $\log_2 \log_2 x$ if $x \geq 4$ and 1 if $x < 4$.)

PROPOSITION 1.1. (**[21, 18]**) *There is a deterministic polynomial-time algorithm that delivers a schedule of makespan*

$$\mathrm{O}((P_{\max} + \Pi_{\max}) \cdot \frac{\log(m\mu)}{\log\log(m\mu)} \cdot \log(\min\{m\mu, p_{\max}\}))$$

for general job-shop scheduling. If we replace m by n in this bound, then such a schedule can also be computed in RNC. This is a ρ-approximation algorithm with $\rho = \mathrm{O}(\log(m\mu)\log(\min\{m\mu, p_{\max}\})/\log\log(m\mu))$. See [21, 9] for further results on approximating some special cases of shop scheduling that are not discussed here.

1.2. Our results. Our first result improves Proposition 1.1 by a doubly logarithmic factor and provides further improvements for important special cases.

THEOREM 1.2. *There are the following deterministic algorithms for general job-shop scheduling, delivering schedules of makespan $\mathrm{O}((P_{\max} + \Pi_{\max}) \cdot \rho)$:*

(a) a polynomial-time algorithm, with

$$\rho = \frac{\log(m\mu)}{\log\log(m\mu)} \cdot \left\lceil \frac{\log(\min\{m\mu, p_{\max}\})}{\log\log(m\mu)} \right\rceil,$$

and if we replace m by n in this bound, then such a schedule can also be computed in NC,

(b) a polynomial-time algorithm, with

$$\rho = \frac{\log m}{\log\log m} \cdot \log(\min\{m\mu, p_{\max}\}), \text{ and}$$

(c) an NC algorithm, with

$$\rho = \frac{\log m}{\log\log m} \cdot \log(\min\{n\mu, p_{\max}\}).$$

Thus, part (a) improves on the previous approximation bound by a doubly logarithmic factor. The impact of parts (b) and (c) is best seen for preemptive scheduling, wherein $p_{\max} = 1$, and for the related situations where $p_{\max}$ is "small". Our motivation for focusing on these cases is twofold. First, preemptability is known to be a powerful primitive in various scheduling models, see, e.g., [4]. Second, the result of Leighton, Maggs and Rao shows that preemptability is powerful for acyclic job-shops (in the sense that there is a schedule of makespan $\mathrm{O}(P_{\max} + \Pi_{\max})$ in the preemptive case). Recall that job-shop scheduling is NP-hard even when $n = m = 3$ and $p_{\max} = 1$. Parts (b) and (c) of Theorem 1.2 show that, as long as the number of machines is small or fixed, we get very good approximations. (It is trivial to get an approximation factor of m: our approximation ratio is $O(\log m/\log\log m)$ if $p_{\max}$ is fixed.) Note that for the case in which $p_{\max}$ is small, part (c) is both a derandomization and an improvement of the previous-best parallel algorithm for job-shop scheduling (see Proposition 1.1).

We further explore the issue of when good approximations are possible, once again with a view to generalizing the result of Leighton, Maggs and Rao [11]; this is done by the somewhat-technical Theorem 1.3. In the statement of the theorem, "with high probability" means "with probability at least $1 - \epsilon$, for a positive constant ϵ. The failure probability ϵ can be made arbitrarily small (exponentially small in the size of the problem instance) by repeating the algorithm many times. Theorem 1.3 shows that if (a) no job requires too much of any given machine for processing, or if (b) repeated uses of the same machine by a given job are well-separated in time, then good approximations are possible. Say that a job-shop instance is *w-separated* if every distinct pair $((M_{j,\ell}, t_{j,\ell}), (M_{j,r}, t_{j,r}))$ of operations of the same job with the same machine (i.e., every pair such that $M_{j,\ell} = M_{j,r}$) has $|\ell - r| \geq w$.

THEOREM 1.3. *There is a randomized polynomial-time algorithm for job-shop scheduling that, with high probability, delivers a schedule of makespan* $\mathrm{O}((P_{\max} + \Pi_{\max}) \cdot \rho)$, *where*

(a) if every job needs at most u time units on each machine then

$$\rho = \frac{\log u}{\log \log u} \cdot \left\lceil \frac{\log(\min\{m\mu, p_{\max}\})}{\log \log u} \right\rceil;$$

(b) if the job-shop instance is w-separated and $p_{\max} = 1$ then

$$\rho = 1 \quad \text{if } w \geq \log(P_{\max} + \Pi_{\max})/2;$$

$$\rho = \frac{\log(P_{\max} + \Pi_{\max})}{w \log(\log(P_{\max} + \Pi_{\max})/w)}, \quad \text{otherwise.}$$

Part (a) of Theorem 1.3 shows quantitatively the advantages of having multiple copies of each machine; in such a case, we can try to spread out the operations of a job somewhat equitably to the various copies. Part (b) of Theorem 1.3 shows that if we have some (limited) flexibility in rearranging the operation sequence of a job, then it may pay to spread out multiple usages of the same machine.

1.3. Main contributions. Most of our results rely on probabilistic ideas; in particular, we exploit a "random delays" technique due to [11]. We make four contributions, which we first sketch in general terms. The rough idea behind the "random delays" technique is as follows. We give each job a delay chosen randomly from a suitable range and independently of the other jobs, and imagine each job waiting out this delay and then running without interruption; next we argue that, with high probability, not too many jobs contend for any given machine at the same time [11, 21]. We then resolve contentions by "expanding" the above "schedule"; the "low contention" property is invoked to argue that a small amount of such expansion suffices. The approach of [21] to this "expansion" problem is as follows. First, they present an upper bound on the maximum amount of contention on any machine at any step, which is shown to hold with high probability. Suppose we are given such a schedule, in which at most s operations contend for any machine at any time. If all operations are of the same length, this can be converted into a valid schedule by an s-fold expansion of each time step. However, the operation lengths may be disparate. But we may round all operation lengths up to the nearest power of two; thus, there will only be $\mathrm{O}(\log p_{\max})$ operation lengths. The approach of [21] is then to carefully decompose the schedule into certain intervals such that within each interval, all operation lengths are the same. These, along with some other ideas, constitute the "expansion" approach of [21].

Our first contribution is a better combinatorial solution to the above expansion problem, which leads to a smaller expansion than that of [21]. In particular, we do not handle different operation lengths separately, but show a way of combining them. The second contribution shows that a relaxed notion of "low contention" suffices: we do not require that the contention on machines be low at each time step. The first contribution helps to prove Theorem 1.2(a); parts (b) and (c) of Theorem 1.2 make use of the second contribution. We de-randomize the sequential formulations using a technique of [2] and then parallelize. A simple but crucial ingredient of Theorem 1.2 is a new way to structure the operations of jobs in an initial (infeasible) schedule; we call this *well-structuredness*, and present it in Section 2. This notion is our third contribution. Finally, Theorem 1.3 comes about by introducing random delays and by using the Lovász Local Lemma (LLL) [7]. Although this is also done in [11], our improvements arise from a study of the correlations involved and by using Theorem 1.2(a). This study of correlations is our fourth contribution. The rest of this paper is organized as follows. Section 2 sets up some preliminary notions, Section 3 presents the proof of Theorem 1.2, and Theorem 1.3 is proved in Section 4.

2. Preliminaries. For any non-negative integer k, we let $[k]$ denote the set $\{1, 2, \ldots, k\}$. The base of the natural logarithm is denoted by e as usual and, for convenience, we may use $exp(x)$ to denote e^x.

As in [21], we assume throughout that all operation lengths are powers of two. This can be achieved by multiplying each operation length by at most two. This assumption on operation lengths will only affect our approximation factor and running time by a constant factor. Thus, $P_{\max}$, $\Pi_{\max}$ and $p_{\max}$ should be replaced by some $P'_{\max} \leq 2P_{\max}$, $\Pi'_{\max} \leq 2\Pi_{\max}$, and $p'_{\max} \leq 2p_{\max}$ respectively, in the sequel. We have avoided using such new notation, to retain simplicity.

2.1. Reductions. It is shown in [21] that, in deterministic polynomial time, we can reduce the general shop-scheduling problem to the case where (i) $p_{\max} \leq n\mu$, and where (ii) $n \leq \mathrm{poly}(m, \mu)$, while incurring an *additive* $\mathrm{O}(P_{\max} + \Pi_{\max})$ term in the makespan of the schedule produced. The reduction (i) also works in NC. (Of the two reductions, (ii) is more involved; it uses, e.g., an algorithm due to [20].)

Thus, for our sequential algorithms we assume that $n \leq \mathrm{poly}(m, \mu)$ and that $p_{\max} \leq \mathrm{poly}(m, \mu)$; while for our NC algorithms we assume only that $p_{\max} \leq n\mu$.

2.2. Bounds. We use the following bounds on the expectation and tails of distributions.

FACT 2.1. [Chernoff, Hoeffding] *Let* $X_1, X_2, \ldots, X_\ell \in [0,1]$ *be* independent *random variables with* $X \doteq \sum_i X_i$. *Then for any* $\delta > 0$, $\mathrm{E}[(1+\delta)^X] \leq e^{\delta \mathrm{E}[X]}$.

We define $G(\mu, \delta) \doteq (e^\delta/(1+\delta)^{1+\delta})^\mu$. Using Markov's inequality and Fact 2.1, we obtain Chernoff and Hoeffding's bounds on the tails of the binomial distribution (see [15]).

FACT 2.2. [Chernoff, Hoeffding] *Let* $X_1, X_2, \ldots, X_\ell \in [0,1]$ *be* independent *random variables with* $X \doteq \sum_i X_i$ *and* $\mathrm{E}[X] = \mu$. *Then for any* $\delta > 0$, $\Pr[X \geq \mu(1+\delta)] \leq G(\mu, \delta)$.

2.3. Random delays. Our algorithms use *random initial delays* which were developed in [11] and used in [21]. A *B-delayed schedule* of a job-shop instance is constructed as follows. Each job J_j is assigned a delay d_j in $\{0, 1, \ldots, B-1\}$. In the resulting B-delayed schedule, the operations of J_j are scheduled consecutively, starting at time d_j. A *random B-delayed schedule* is a B-delayed schedule in which the delays have been chosen independently and uniformly at random from $\{0, 1, \ldots, B-1\}$. Our algorithms schedule a job-shop instance by choosing a random B-delayed schedule for some suitable B, and then expanding this schedule to resolve conflicts between operations that use the same machine at the same time.

For a B-delayed schedule $\mathcal{S}$, the *contention*, $C(M_i, t)$, is the number of operations scheduled on machine M_i in the time interval $[t, t+1)$. (Recall that operation lengths are integral.) For any job J_j, define the random variable $X_{i,j,t}$ to be 1 if some operation of J_j is scheduled on M_i in the time interval $[t, t+1)$ by $\mathcal{S}$, and 0 otherwise. Since no two operations of J_j contend for M_i simultaneously, $C(M_i, t) = \sum_j X_{i,j,t}$. If the delays are chosen uniformly at random and $B \geq \Pi_{\max}$, then $\mathrm{E}[X_{i,j,t}]$ is at most the total processing time of J_j on M_i divided by $\Pi_{\max}$. Thus, $\mathrm{E}[C(M_i, t)] = \sum_j \mathrm{E}[X_{i,j,t}] \leq \Pi_{\max}/\Pi_{\max} = 1$. We also note that the random variables $\{X_{i,j,t} \mid j \in [n]\}$ are mutually independent, for any given i and t. We record all this as follows.

FACT 2.3. *If $B \geq \Pi_{\max}$ and $\mathcal{S}$ is a random B-delayed schedule then for any machine M_i and any time t, $C(M_i, t) = \sum_j X_{i,j,t}$, where the 0-1 random variables $\{X_{i,j,t} \mid j \in [n]\}$ are mutually independent. Also,* $\mathrm{E}[C(M_i, t)] \leq 1$.

2.4. Well-structuredness. Recall that all operation lengths are assumed to be powers of two. We say that a delayed schedule $\mathcal{S}$ is *well-structured* if for each k, all operations with length 2^k begin in $\mathcal{S}$ at a time instant that is an integral multiple of 2^k. We shall use the following simple way of constructing such schedules from randomly delayed schedules. First create a new job-shop instance by replacing each operation $(M_{j,\ell}, t_{j,\ell})$ by the operation $(M_{j,\ell}, 2 \cdot t_{j,\ell})$. Suppose $\mathcal{S}$ is a random B-delayed schedule for this modified instance, for some B; we will call $\mathcal{S}$ a *padded random B-delayed schedule*. From $\mathcal{S}$, we can construct a well-structured delayed schedule, $\mathcal{S}'$, for the original job-shop instance: simply insert $(M_{j,l}, t_{j,l})$ with the correct boundary in the slot assigned to $(M_{j,l}, 2 \cdot t_{j,l})$ by $\mathcal{S}$. S' will be called a *well-structured random B-delayed schedule* for the original job-shop instance.

3. Proof of Theorem 1.2. In this section we prove Theorem 1.2. In Section 3.1 we give a randomized polynomial-time algorithm that proves part (b) of the theorem. In Section 3.2 we improve the algorithm to prove part (a). Finally we discuss the derandomization and parallelization of these algorithms in Section 3.3. Throughout, we shall assume upper bounds on n and $p_{\max}$ as described in Section 2.1; this explains terms such as $\log(\min\{m\mu, p_{\max}\})$ in the bounds of Theorem 1.2. Given a delayed schedule $\mathcal{S}$, define $C(t) \doteq \max_i \; C(M_i, t)$.

LEMMA 3.1. *There is a randomized polynomial-time algorithm that takes a job-shop instance and produces a* well-structured *delayed schedule which has a makespan $L \leq 2(P_{\max} + \Pi_{\max})$. With high probability, this schedule satisfies:*

(a) $\forall i \in [m]\ \forall t \in \{0, 1, \ldots, L-1\}$, $C(M_i, t) \leq \alpha$, and

(b) $\sum_{t=0}^{L-1} C(t) \leq \beta(P_{\max} + \Pi_{\max})$,

where $\alpha = c_1 \log(m\mu)/\log\log(m\mu)$ and $\beta = c_2 \log m/\log\log m$, for sufficiently large constants $c_1, c_2 > 0$.

Proof. Recall that all operation lengths are assumed to be powers of 2. Let $B = 2\Pi_{\max}$ and let $\mathcal{S}$ be a *padded* random B-delayed schedule of the new instance, as described in Section 2.4. $\mathcal{S}$ has a makespan of at most $2(P_{\max} + \Pi_{\max})$. Let $\mathcal{S}'$ be the well-structured random B-delayed schedule for the original instance that can be constructed from $\mathcal{S}$, as described in Section 2.4. The contention on any machine at any time under $\mathcal{S}'$ is clearly no more than under $\mathcal{S}$. Thus in order to show that $\mathcal{S}'$ satisfies (a) and (b) with high probability, it suffices to show that $\mathcal{S}$ has this property. We will prove this now.

Part (a). The following proof is based on that of [21]. Fix any positive integer k, and any M_i. For any set $U = \{u_1, u_2, \ldots, u_k\}$ of k units of processing that need to be done on M_i, let Collide(U) be the event that in S all these k units get scheduled at the same unit of time on M_i. It is not hard to see that $\Pr[\text{Collide}(U)] \leq (1/B)^{k-1}$. (If $u_1, \ldots, u_k$ are from different jobs then $\Pr[\text{Collide}(U)] \leq (1/B)^{k-1}$. Otherwise, $\Pr[\text{Collide}(U)] = 0$.) Recall that $B = 2\Pi_{\max}$. Since there are at most $\binom{2\Pi_{\max}}{k}$ ways of choosing U, we get

$$\Pr[\exists t: \ C(M_i, t) \geq k] = \Pr[\exists U: \ \text{Collide}(U)] \leq \binom{2\Pi_{\max}}{k} (1/(2\Pi_{\max}))^{k-1},$$

and so $\Pr[\exists t: \ C(M_i, t) \geq k] \leq 2\Pi_{\max}/k!$. Thus,

$$\Pr[\exists t \ \exists i: \ C(M_i, t) \geq k] \leq 2m\Pi_{\max}/k!.$$

But $\Pi_{\max} \leq n\mu p_{\max}$, which by our assumptions in Section 2.1 is poly(m, μ). Since $\lceil\alpha\rceil! > (m\mu)^{c_1/2}$ for sufficiently large m or μ, we can satisfy (a) with high probability if we choose c_1 sufficiently large.

Part (b). Let $\gamma = \beta\epsilon/2$, where ϵ is the desired constant in the probability bound. Let the constant c_2 in the definition of β be sufficiently large so that $\gamma > 1$. Fix any M_i and t, and let $\lambda = \mathrm{E}[C(M_i, t)]$. (By Fact 2.3, $\lambda \leq 1$.) By Fact 2.1, with $1 + \delta = \gamma$,

$$\mathrm{E}[\gamma^{C(M_i,t)}] \leq e^{(\gamma-1)\lambda} \leq e^{(\gamma-1)}.$$

Hence, for any given t,

$$\mathrm{E}[\gamma^{C(t)}] = E[\gamma^{\max_{i\in[m]} C(M_i,t)}] \leq E[\sum_{i\in[m]} \gamma^{C(M_i,t)}] = \sum_{i\in[m]} \mathrm{E}[\gamma^{C(M_i,t)}] \leq me^{\gamma-1}. \tag{3.1}$$

Since the function $x \mapsto \gamma^x$ is convex, by Jensen's inequality we get that $\mathrm{E}[\gamma^{C(t)}] \geq \gamma^{\mathrm{E}[C(t)]}$. If we choose c_2 sufficiently large then $\gamma^\gamma \geq me^{\gamma-1}$. Combining these observations with bound (3.1), we get $\mathrm{E}[C(t)] \leq \gamma$. By linearity of expectation, $\mathrm{E}[\sum_t C(t)] \leq 2\gamma(P_{\max} + \Pi_{\max})$ and finally, by Markov's inequality, we have

$$\Pr[\sum_t C(t) > \beta(P_{\max} + \Pi_{\max})] \leq 2\gamma/\beta = \epsilon.$$

□

3.1. Proof of Theorem 1.2(b). Recall that our goal is a polynomial-time algorithm which delivers a schedule with makespan $\mathrm{O}((P_{\max} + \Pi_{\max}) \cdot \frac{\log m}{\log\log m} \cdot \log(\min\{m\mu, p_{\max}\}))$. Assume $\mathcal{S}$ is a delayed schedule satisfying the conditions of Lemma 3.1 with makespan $L = \mathrm{O}(P_{\max} + \Pi_{\max})$. We begin by partitioning the schedule into *frames*, i.e., time intervals $\{[ip_{\max}, (i+1)p_{\max}), i = 0, 1, \ldots, \lceil L/p_{\max}\rceil - 1\}$. By the definition of $p_{\max}$ and the fact that $\mathcal{S}$ is well-structured, no operation straddles a frame. For example, see Figure 3.1.

We construct a feasible schedule for the operations performed under schedule $\mathcal{S}$ for each frame. Concatenating these schedules yields a feasible schedule for the original problem. We give the frame-scheduling algorithm where, without loss of generality, we assume that its input is the first frame.

Let T be a rooted complete binary tree with $p_{\max}$ leaves. For every node u of T, let $l(u)$ and $r(u)$ be the labels, respectively, of the leftmost and rightmost leaves of the subtree rooted at u. We shall associate the operations scheduled during the frame with the nodes of T in a natural way. For $i = 1, \ldots, m$ we define $S_i(u)$ to be those operations that are scheduled on M_i by $\mathcal{S}$ for precisely the time interval $[l(u), r(u) + 1)$; each operation scheduled by $\mathcal{S}$ in the first frame is in exactly one $S_i(u)$. For example, see Figure 3.2. Let $p(u) = (r(u) - l(u) + 1) \cdot \max_i ||S_i(u)||$, where $||S_i(u)||$ denotes the cardinality of $S_i(u)$. $p(u)$ is the amount of time needed to perform the operations associated with u. For example, see Figure 3.3. Let the nodes of T be numbered as $u_1, u_2, \ldots$ in the *preorder* traversal of T. Define $f(u_1) = 0$ and for $j \geq 2$, let $f(u_j) = \sum_{k<j} p(u_k)$. For example, see Figure 3.4. The algorithm simply schedules the operations in $S_i(u)$ on machine M_i consecutively beginning at time $f(u) + 1$ and concluding by the end of timestep $f(u) + p(u)$. Let $\mathcal{S}'$ be the resulting schedule. For example, see Figure 3.5. Note that our algorithm does not necessarily give the same schedule as the algorithm of Shmoys, Stein and Wein. For instance, our algorithm produces a different schedule than the one that their algorithm produces on the example given in [21]. Part (b) of Theorem 1.2 follows from Lemma 3.1 and the following lemma.

LEMMA 3.2. *$\mathcal{S}'$ is feasible and has makespan at most $\sum_{u \in T} p(u)$, which is at most $(1 + \log_2 p_{\max}) \cdot \sum_{j=0}^{p_{\max}-1} C(j)$, where $C(t)$ is the maximum contention at time t under schedule S.*

Proof. By construction, no machine performs more than one operation at a time. Suppose O_1 and O_2 are distinct operations of job J scheduled in the first frame. Assume $O_1 \in S_i(u)$ and $O_2 \in S_j(v)$, where possibly $i = j$. Assume O_1 concludes before O_2 begins under $\mathcal{S}$; thus u and v are roots of disjoint subtrees of T and u precedes v in the preorder traversal of T. Thus O_1 concludes before O_2 begins in $\mathcal{S}'$ and the new schedule is feasible.

Clearly the makespan of $\mathcal{S}'$ is at most $\sum_{u \in T} p(u)$. Fix a node u at some height k in T. (We take leaves to have height 0.) Then $p(u) = 2^k \max_i ||S_i(u)||$. Since the maximum number of jobs scheduled at any time t on any machine under $\mathcal{S}$ is $C(t)$, we get that $\forall t \in [l(u), \ldots, r(u)]$, $\max_i ||S_i(u)|| \leq C(t)$. Thus,

$$p(u) \leq 2^k \max_i ||S_i(u)|| \leq \sum_{t \in [l(u), \ldots, r(u)]} C(t).$$

Since each leaf of T has $(1 + \log_2 p_{\max})$ ancestors, the makespan of $\mathcal{S}'$ is at most

$$\sum_{u \in T} p(u) \leq \sum_{u \in T} \sum_{t \in [l(u), \ldots, r(u)]} C(t) = (1 + \log_2 p_{\max}) \cdot \sum_{t=0}^{p_{\max}-1} C(t).$$

□

3.2. Proof of Theorem 1.2(a). Recall that our goal is a polynomial-time algorithm which delivers a schedule with makespan $\mathrm{O}((P_{\max} + \Pi_{\max}) \cdot \frac{\log(m\mu)}{\log\log(m\mu)} \cdot \left\lceil \frac{\log(\min\{m\mu, p_{\max}\})}{\log\log(m\mu)} \right\rceil)$. We give a slightly different frame-scheduling algorithm and show that the feasible schedule for each frame has makespan $\mathrm{O}(p_{\max}\alpha \lceil \log(p_{\max})/\log \alpha \rceil)$, where $\alpha = c_1 \log(m\mu)/\log\log(m\mu)$ as in Lemma 3.1. Without loss of generality, we assume that α is a power of 2 (by increasing it if necessary). Thus, under the assumptions from Section 2.1, the final schedule satisfies the bounds of Theorem 1.2(a).

The difficulty with the algorithm given in Section 3.1 is that the operations may be badly distributed to the nodes of T by $\mathcal{S}$ which would make $\mathcal{S}'$ inefficient. To clarify, consider the example given in Figures 3.1–3.5. In this case, node u_{10} is assigned operations C and K and node u_{11} is assigned operation H. The algorithm schedules operations C and K before operation H. However, since H is on a different machine from C and K, it could have been scheduled to overlap C or K. In this section, we show how to overcome this problem by "pushing down" operations C and K to nodes u_{11} and u_{12}.

The algorithm that we describe here starts with the allocation of operations to nodes of T that is defined in Section 3.1. That is, $S_i(u)$ is taken to be the set of operations that are scheduled on M_i by S for time interval $[l(u), r(u)+1)$. The algorithm then chops T into disjoint subtrees in a manner described below. For each subtree, it re-distributes the operations that are allocated to the nodes of the subtree by "pushing" some operations from parents to children (in a manner which will be described shortly). After the re-distribution, $R_i(u)$ is the set of machine-i operations that are allocated to node u. $p(u)$ is then taken to be the maximum over all i, of the sum of the lengths of the operations in $R_i(u)$. The algorithm then finishes the algorithm of Section 3.1: the p-values computed for each node are used to compute $f(v)$ (for every node v). Then the operations in $R_i(v)$ are scheduled beginning at time $f(v)+1$ and concluding by the end of timestep $f(v)+p(v)$.

The partitioning of T is done by removing all edges from parents with height equal to 0 modulo $\log\alpha$. (Thus, every resulting subtree T' has height at most $\log\alpha$.)

Let lg denote the logarithm to the base 2. (In some places below, we will not be able to use $\log x$ since, as defined by us, $\log x$ does not always equal the logarithm of x to the base 2. So we need lg.)

The re-distribution of operations for subtree T' proceeds in a top-down manner, independently for each machine M_i. We will illustrate the process with the job-shop instance in Figure 3.6, where we assume (for descriptive purposes) that T has only one sub-tree T'. Start at the root, u_1, of T'. Suppose that u_1 has h operations allocated to it. (In this case, $h=3$.) Let $h' = 2^{\lceil \lg h \rceil}$ (in this case, $h'=4$) and allocate $h'-h$ dummy operations $\emptyset$ to T' as in Figure 3.7. (The reason for adding the dummy operations is to make the number of operations at the root equal to a power of 2.) If the height of the subtree rooted at u_1 (in this case, 2) is at least $\lg(h')$ (which is also 2 in this case), then the h' operations originally allocated to u_1 are re-allocated to the h' nodes that are at distance $\lg(h')$ below u as in Figure 3.8. Next, the operations are further re-allocated recursively in the subtrees below u_1 (in this case, the operations are recursively re-allocated in the subtrees rooted at u_2 and u_5). If, in one of these recursive calls, the height, k, of the subtree being considered is less than $\lg(h')$ (where h' is the number of *originally allocated* operations at the root, counting dummy operations) then $h'/2^k$ of the operations originally allocated to the root are re-allocated to each of the leaves. For example, in the recursive call on the subtree rooted at u_2 in Figure 3.8, $h'=4$ (because a dummy operation is added to u_2 to make the number of operations a power of 2) and the height, k, of the subtree below u_2 is 1. Thus, $h'/2^1$ operations are pushed from u_2 to each of its children as in Figure 3.9. The recursive call at u_5 and the recursive calls at the leaves do not further re-distribute operations.

A more formal description of the pushdown algorithm is as follows. As above, we assume that $||S_i(v)||$ is a power of two for all i and v; furthermore, although we will push some operations down the tree, $S_i(v)$ will throughout refer to the *original* set of operations scheduled on M_i for the time interval $[l(v), r(v)+1)$. First partition the tree T into disjoint subtrees, by removing all edges from parents with height equal to 0 modulo $\log\alpha$. We then proceed independently for each subtree T' that is produced from the partition, and for each machine M_i, by calling a recursive procedure pushdown(T', i), which we describe now. Given a binary tree T'' with root u and a machine index i, pushdown(T'', i) is as follows. If T'' is a leaf, the procedure does nothing. Otherwise, suppose $||S_i(u)|| = h'$, with h' being a power of two. If the height k of T'' is at least $\lg(h')$, then the h' operations of $S_i(u)$ are re-allocated to the h' nodes that are at distance $\lg(h')$ below u; else if $k < \lg(h')$, then $h'/2^k$ of the operations in $S_i(u)$ are re-allocated to each of the leaves of T''. Finally, we recursively call the procedure on the left and right subtrees of T''.

Note that if the new algorithm is applied to the problem instance from Figures 3.1–3.5 then the makespan is reduced by one, because operations C and K are pushed down to the leaves so operation H is scheduled to overlap operation C.

Let $\mathcal{S}'$ denote the schedule produced (from S) by the new algorithm.

LEMMA 3.3. *$\mathcal{S}'$ is a feasible schedule with makespan* $\mathrm{O}(p_{\max}\alpha\lceil\log p_{\max}/\log\alpha\rceil)$.

Proof. The proof that $\mathcal{S}'$ is feasible follows exactly as before. The makespan of $\mathcal{S}'$ is no more than $\sum_{u\in T} p(u)$.

Consider a subtree T' of the partition. Assume the leaves of T' are at height j in T. Let w be a node in T' and let V be the subset of nodes of T' consisting of w and its ancestors in T'.

First suppose w is a leaf. Let v be a node in V and assume that v has height k in T' with $||S_i(v)|| = h$. (See Figure 3.10.)

Then v contributes at most $2^{\lceil\lg h\rceil}/2^k$ operations to $R_i(w)$ and each has length 2^{j+k}. The time needed to perform these operations is $2^{\lceil\lg h\rceil-k}\cdot 2^{j+k} = 2^{\lceil\lg h\rceil+j}$. By Lemma 3.1, part (a), $\sum_{v\in V}||S_i(v)|| \le 2\alpha$. (The factor of 2 arises from the (possible) padding of $S_i(v)$ with dummy operations.) Thus $p(w) \le 2^{j+1}\alpha$.

Now suppose w is at height $r > 0$ in T'. (See Figure 3.11.) A node $v \in V$ at height $r+k$ in T' contributes at most one operation to $R_i(w)$ and its length is 2^{j+k+r}. Thus $p(w) \le \sum_{k=0}^{\log\alpha - r} 2^{j+k+r} \le 2^{j+1}\alpha$.

Thus, if node w is at height $r+j$ in T and is in the layer of the partition containing T', then $p(w) \le 2^{j+1}\alpha$; also, there are $p_{\max}/2^{r+j}$ nodes at this height in T. The sum of these $p(w)$'s is thus at most $2\alpha p_{\max}/2^r$. Each layer therefore contributes at most $4\alpha p_{\max}$, and there are $\lceil(\log p_{\max})/(\log\alpha)\rceil$ layers. Thus $\sum_{v\in T} p(v)$ satisfies the bound of the lemma. ◻

3.3. Derandomization and parallelization. Note that all portions of our algorithm are deterministic (and can be implemented in NC), except for the setting of the initial random delays, which we show how to derandomize now. The method of conditional probabilities could be applied to give the sequential derandomization, however that result will follow from the NC algorithm that we present. We begin with a technical lemma.

LEMMA 3.4. *Let $x_1, x_2, \ldots, x_\ell$ be non-negative integers such that $\sum_i x_i = \ell a$, for some $a \ge 1$. Let $k \le a$ be any positive integer. Then, $\sum_{i=1}^{\ell}\binom{x_i}{k} \ge \ell\cdot\binom{a}{k}$.*

Proof. For real x, we define, as usual, $\binom{x}{k} \doteq (x(x-1)\cdots(x-k+1))/k!$. We first verify that the function $f(x) = \binom{x}{k}$ is non-decreasing and convex for $x \ge k$, by a simple check that the first and second derivatives of f are non-negative for $x \ge k$. Think of minimizing $\sum_i\binom{x_i}{k}$ subject to the given constraints. If $x_i \le (k-1)$ for some i, then there should be an index j such that $x_j \ge (k+1)$, since $\sum_i x_i \ge \ell k$. Thus, we can lessen the objective function by simultaneously setting $x_i := x_i + 1$ and

$x_j := x_j - 1$. Hence we may assume that all the integers x_i are at least k. By the convexity of f for $x \geq k$, we see that the objective function is at least $\sum_{i=1}^{\ell} \binom{a}{k}$. □

Define, for $z = (z_1, z_2, \ldots, z_n) \in \Re^n$, a family of symmetric polynomials $S_j(z), j = 0, 1, \ldots, n$, where $S_0(z) \equiv 1$, and for $1 \leq j \leq n$, $S_j(z) \doteq \sum_{1 \leq i_1 < i_2 \cdots < i_j \leq n} z_{i_1} z_{i_2} \cdots z_{i_j}$. We recall one of the main results of [2] (this is not explicitly presented in [2], but is an obvious corollary of the results of Section 4 in [2]). In the statement of Proposition 3.5, the function G refers to the one introduced in Section 2.2. Namely, $G(\mu, \delta) = (e^{\delta}/(1+\delta)^{1+\delta})^{\mu}$.

PROPOSITION 3.5. **([2])** *Suppose we are given m* independent *random variables $y_1, \ldots, y_m$, each of which takes values* uniformly *in $R = \{0, 1, \ldots, 2^b - 1\}$ where $b = \mathrm{O}(\log N)$; N here is a parameter that roughly stands for "input length", and $m = N^{\mathrm{O}(1)}$. Suppose we are also given, for each $j \in [m]$, a finite set of binary random variables $\{z_{jt} : t = 1, 2, \ldots\}$ where z_{jt} is 1 if and only if y_j lies in some fixed subset R_{jt} of R. Also given are r random variables*

$$U_i = \sum_{j=1}^{m} z_{j,f(i,j)}, \ i \in [r],$$

where f is some arbitrary given function. If $\mathrm{E}[U_i] < 1$ for each i, then given any positive integer k such that $k = \mathrm{O}(\log N)$, we can find, deterministically using $N^{\mathrm{O}(1)}$ processors and $\mathrm{O}(\log^{\mathrm{O}(1)} N)$ time on the EREW PRAM, a setting $y_1 := w_1, \ldots, y_m := w_m$ such that

$$\sum_{i \in [r]} S_k(z_{1,f(i,1)}, \ldots, z_{m,f(i,m)}) \leq rG(1, k-1)(1 + N^{-c}),$$

for any desired constant $c > 0$.

In our setting, the random variables y_i are the initial random delays of the jobs. It is easy to verify that each random variable $C(M_i, t)$ is of the form of some U_j in the notation of Proposition 3.5. By giving the initial random delays in the range $\{0, 1, \ldots, 2\Pi_{\max}\}$ instead of from $\{0, 1, \ldots, 2\Pi_{\max} - 1\}$, we can ensure the condition $\mathrm{E}[U_j] < 1$ of Proposition 3.5 ($\mathrm{E}[C(M_i, t)] \leq 2\Pi_{\max}/(2\Pi_{\max} + 1)$ now). Let α and β be as in Lemma 3.1, and note that both are logarithmically bounded in the length of the input, as required for the parameter k in Proposition 3.5. Let the random variables $X_{i,j,t}$ be as in Fact 2.3. From the proof of part (a) of Lemma 3.1, we see that $\sum_{i,t} G(1, \alpha - 1)$ is smaller than 1; thus, by Proposition 3.5, we can find a setting $\vec{w}$ for the initial delays in NC such that

$$\sum_{i,t} S_{\alpha}(X_{i,1,t}, X_{i,2,t}, \ldots, X_{i,n,t}) < 1. \tag{3.2}$$

If the congestion of some machine M_i at some t were at least α due to the above setting of the initial delays to $\vec{w}$, then the left-hand-side of (3.2) would be at least 1, contradicting (3.2). Thus, we have an NC derandomization of Theorem 1.2(a).

As for Theorem 1.2(b), we can similarly find an NC assignment of initial delays $\vec{w}$ such that

$$\begin{aligned} \sum_{i,t} S_{\beta}(X_{i,1,t}, X_{i,2,t}, \ldots, X_{i,n,t}) &= \mathrm{O}((P_{\max} + \Pi_{\max}) m G(1, \beta - 1)) \\ &= \mathrm{O}((P_{\max} + \Pi_{\max})). \end{aligned} \tag{3.3}$$

Let $C(t)$ be the (deterministic) maximum contention at time t, due to this setting. Note that

$$\binom{C(t)}{\beta} \leq \sum_i S_\beta(X_{i,1,t}, X_{i,2,t}, \ldots, X_{i,n,t}).$$

Thus, by (3.3), we see that

$$\sum_t \binom{C(t)}{\beta} = \mathrm{O}((P_{\max} + \Pi_{\max})).$$

We invoke Lemma 3.4 to conclude that $\sum_t C(t) = \mathrm{O}((P_{\max} + \Pi_{\max})\beta)$; thus, we have an NC derandomization of Theorem 1.2(b).

We remark that the work of Mahajan, Ramos & Subrahmanyam [13] could also be used to obtain an NC derandomization.

4. Proof of Theorem 1.3. We now set about to prove Theorem 1.3; we are very much motivated here by the framework of [5, 1, 12] and of Section 6.1 of [17]. The new ideas we need are due to the two basic ways in which job-shop scheduling generalizes packet routing: both acyclicity and the "$p_{\max} = 1$" condition can be violated. Theorem 4.3 is used first (in the next subsection) and proved later; this is to help the reader get to some of the new ideas quickly. The algorithms are shown in Sections 4.2 and 4.3.

4.1. Preliminary results. We start with a standard fact about the function G of Section 2.2, where $G(\mu, \delta) = (e^\delta/(1+\delta)^{1+\delta})^\mu$.

FACT 4.1. *(a) If $\delta \in [0,1]$, then $e^\delta/(1+\delta)^{(1+\delta)} \leq e^{-\delta^2/3}$. (b) If $0 < \mu_1 \leq \mu_2$, then for any $\delta \geq 0$, $G(\mu_1, \mu_2\delta/\mu_1) \leq G(\mu_2, \delta)$.*

Proof. (a) The proof follows from observing that the function $\delta \mapsto \ln(e^{-\delta^2/3}(1+\delta)^{(1+\delta)}e^{-\delta})$ is 0 when $\delta = 0$, and that its derivative is $\ln(1+\delta) - 2\delta/3$ which is non-negative for $\delta \in [0,1]$.

(b) We need to show that

$$(1+\delta)^{(1+\delta)\mu_2} \leq \left(1 + \frac{\mu_2\delta}{\mu_1}\right)^{(1+\frac{\mu_2\delta}{\mu_1})\mu_1},$$

i.e., that $\Phi(v) \doteq (1+v\delta)\ln(1+v\delta) - v(1+\delta)\ln(1+\delta) \geq 0$ for all $v \geq 1$. We have $\Phi(1) = 0$; $\Phi'(v) = \delta + \delta\ln(1+v\delta) - (1+\delta)\ln(1+\delta)$. For $v \geq 1$, $\Phi'(v) \geq \delta - \ln(1+\delta) \geq 0$. ☐

The next lemma follows from [18].

LEMMA 4.2. **([18])** *Let $X_1, \ldots, X_\ell \in \{0,1\}$ be random variables such that, for any set $T \subseteq \{1, 2, \ldots, \ell\}$, $\Pr[\bigwedge_{i \in T}(X_i = 1)] \leq \prod_{i \in T} \Pr[X_i = 1]$; informally, the X_i are "negatively correlated". Then if $X = \sum_i X_i$ with $\mathrm{E}[X] \leq \mu$, we have, for any $\delta \geq 0$, $\Pr[X \geq \mu(1+\delta)] \leq G(\mu, \delta)$.*

Suppose we are given a job-shop instance I. A *delayed schedule* $\mathcal{S}$ for I is any "schedule" in which each job J_j waits for some arbitrary non-negative integral amount of time d_j, and then gets processed continuously. (Thus, $\mathcal{S}$ is a delayed schedule if and only if there exists some non-negative integer B such that $\mathcal{S}$ is a B-delayed schedule.) Suppose, for some non-negative integer B', we choose integers $d'_1, d'_2, \ldots, d'_j$ uniformly at random and independently, from $\{0, 1, \ldots, B'-1\}$. The (random) schedule obtained by giving an initial delay of d'_j (**in addition to** the d_j above) to each job J_j, will be called a *random* $(B', \mathcal{S})$*-delayed schedule.* Note that this also will be a delayed schedule.

We require a few more definitions related to $\mathcal{S}$ as above. Suppose L denotes the makespan of $\mathcal{S}$. Then, given an integer ℓ, an ℓ*-interval* is any time interval of the form $[t, t+\ell)$, where t is an integer such that $0 \le t \le L-1$. We denote the interval $[t, t+\ell)$ by F_t. The *contention of machine M_i in interval F_k in the schedule* $\mathcal{S}$, denoted $C_{\mathcal{S},\ell}(i,k)$, is the total processing time on M_i within F_k, in the schedule $\mathcal{S}$. (Suppose, for instance, an operation O of length $\ell+2$ uses M_i and is scheduled to run on M_i in the interval $[t, t+\ell+2)$, in $\mathcal{S}$. Then, for example, O contributes a value of ℓ to $C_{\mathcal{S},\ell}(i,t)$ and a value of three to $C_{\mathcal{S},\ell}(i, t+\ell-1)$.)

Given any integers j_1, j_2 such that $1 \le j_1 \le j_2 \le n$, we let $C'_{\mathcal{S},\ell}(i,k,j_1,j_2)$ denote the total processing time on machine M_i in the interval F_k in the schedule S that is imposed by jobs $J_{j_1}, J_{j_1+1}, \ldots, J_{j_2}$. (In particular, $C'_{\mathcal{S},\ell}(i,k,1,n) = C_{\mathcal{S},\ell}(i,k)$.)

Given a *delayed* schedule $\mathcal{S}$, we call $\mathcal{S}$ an (L,ℓ,C)-schedule if and only if:

- the makespan of $\mathcal{S}$ is at most L, and
- for all machines M_i and all ℓ-intervals F_k, $C_{\mathcal{S},\ell}(i,k) \le C$.

We emphasize that this notation will be employed only for delayed schedules.

We start with Theorem 4.3, which will be of much help in proving Theorem 1.3. Given an (L,ℓ,C_0)-schedule for a job-shop instance, Theorem 4.3 shows a sufficient condition under which we can efficiently construct an $(L+B,\ell',C_1)$-schedule for appropriate values of B, ℓ' and C_1. In most of our applications of the theorem, we will have: (i) $\ell' \ll \ell$, (ii) $B \ll L$, and (iii) C_1 sufficiently small so that the new "relative congestion" C_1/ℓ' is not much more than the original relative congestion C_0/ℓ. Thus, by slightly increasing the makespan of the delayed schedule, we are able to bound the relative congestion in intervals of much smaller length (note from (i) that $\ell' \ll \ell$). Appropriate repetitions of this idea, along with some other tools, will help us prove Theorem 1.3.

For convenience, we define $x^+ = \max(x, 0)$.

THEOREM 4.3. *There is a sufficiently large constant $c_3 > 0$ such that the following holds. Suppose $\mathcal{S}$ is an (L,ℓ,C)-schedule for a given job-shop instance I, for some L,ℓ,C. Let non-negative integers $\ell' \le \ell$ and $B \le \ell - \ell' + 1$ be arbitrary; let $\mathcal{S}'$ be a random $(B,\mathcal{S})$-delayed schedule.*

Suppose $\delta > 0$ is such that for all integers $i \in [m]$, $0 \le k \le L+B-1$ and $1 \le j_1 \le j_2 \le n$,

$$\Pr[C'_{\mathcal{S}',\ell'}(i,k,j_1,j_2) \ge \frac{\ell'}{B} \cdot (C'_{\mathcal{S},\ell}(i,(k-B+1)^+,j_1,j_2) + C\delta)] \le (\max\{L,B,C\})^{-c_3}.$$

Then there is a Las Vegas algorithm to construct an $(L+B, \ell', \frac{C\ell'}{B}\cdot(1+3\delta))$-schedule for I; the expected running time of the algorithm is poly(m,L,ℓ,C). The proof of Theorem 4.3 will be presented in Section 4.5. Using ideas from our earlier proofs, we obtain the following corollary.

COROLLARY 4.4. *For general job-shop scheduling, there is a polynomial-time Las Vegas algorithm to construct a schedule of makespan*

$$\mathrm{O}\left((P_{\max}+\Pi_{\max}) \cdot \frac{\log(P_{\max}+\Pi_{\max})}{\log\log(P_{\max}+\Pi_{\max})} \cdot \left\lceil \frac{\log(\min\{m\mu, p_{\max}\})}{\log\log(P_{\max}+\Pi_{\max})} \right\rceil\right).$$

Proof. Let I be a job-shop scheduling instance with associated values $P_{\max}$ and $\Pi_{\max}$; define $L = 2P_{\max}$ and $B = 2\Pi_{\max}$. Let I' be the modified instance formed by replacing each operation $(M_{j,k}, t_{j,k})$ by the operation $(M_{j,k}, 2 \cdot t_{j,k})$. We trivially have an (L, B, B)-schedule $\mathcal{S}$ for I'. Choose $\lambda = c' \log(P_{\max} + \Pi_{\max}) / \log\log(P_{\max} + \Pi_{\max})$; c' is a suitably large constant as specified below.

Let $\mathcal{S}'$ denote the random $(B, \mathcal{S})$-delayed schedule. From the proof of Lemma 3.1 (a), we find that $\Pr[C'_{\mathcal{S}',1}(i,k,j_1,j_2) \geq \lambda] \leq (\max\{L,B,C\})^{-c_3}$ will hold for all i,k,j_1,j_2, by making c' large. Thus, by setting $\ell' = 1$ in Theorem 4.3, we can efficiently find an $(L{+}B, 1, C_0)$-schedule for I', where $C_0 = \mathrm{O}(\lambda)$. So we can efficiently construct a well-structured schedule for I with makespan $\mathrm{O}(P_{\max} + \Pi_{\max})$ in which, for all machines M_i and time steps t, the number of operations scheduled on machine M_i in the time interval $[t, t+1)$ is at most $\mathrm{O}(\lambda)$. The corollary now follows from the proof of Theorem 1.2(a) by using this fact in place of Lemma 3.1. □

We now present Lemma 4.5, which shows a way of using Theorem 4.3. The notion of "w-separated" in its part (b), is as defined in Section 1.2. Namely, every distinct pair of operations of the same job with the same machine has at least $w-1$ operations between them.

LEMMA 4.5. *(a) Consider any job-shop instance I in which any job needs at most u units of processing on any machine. Suppose $\mathcal{S}$ is some (L,ℓ,C)-schedule for I. For non-negative integers $\ell' \leq \ell$ and $B \leq \ell - \ell' + 1$, suppose $\mathcal{S}'$ denotes the random $(B,\mathcal{S})$-delayed schedule. Then, for any $\delta > 0$ and all i,k,j_1,j_2,*

$$\Pr[C'_{\mathcal{S}',\ell'}(i,k,j_1,j_2) \geq \frac{\ell'}{B} \cdot (C'_{\mathcal{S},\ell}(i,(k-B+1)^+,j_1,j_2) + C\delta)] \leq G(C\ell'/(Bu),\delta).$$

(b) Suppose I is a w-separated job-shop instance with $p_{\max} = 1$, and that $\mathcal{S}$ denotes the (unique) 0-delayed schedule for I. Let $\mathcal{S}'$ denote the random $\Pi_{\max}$-delayed schedule for I. Then, for any $\delta > 0$ and all i,k,j_1,j_2,

$$\Pr[C'_{\mathcal{S}',w}(i,k,j_1,j_2) \geq \frac{w}{\Pi_{\max}} \cdot (C'_{\mathcal{S},\Pi_{\max}+w-1}(i,(k-\Pi_{\max}+1)^+,j_1,j_2) + \Pi_{\max}\delta)] \leq G(w,\delta).$$

Proof. To have some common notation for parts (a) and (b), we define the following quantities for (b). First, in (b), S is a $(P_{\max},\ell,C)$-schedule, where, e.g., $\ell = \Pi_{\max} + w - 1$ and $C = \Pi_{\max}$. Also, in (b), $\mathcal{S}'$ is the random $(B,\mathcal{S})$-delayed schedule, where $B = \Pi_{\max}$; we also set $\ell' = w$ in (b). Note that the conditions $\ell' \leq \ell$ and $B \leq \ell - \ell' + 1$ now hold for (b) also.

We now make some observations common to (a) and (b). Fix i,k,j_1,j_2. Since all new delays introduced by $\mathcal{S}'$ lie in $\{0,1,\ldots,B-1\}$, the only units of processing that can get scheduled on M_i in the interval $[k,k+\ell')$ in $\mathcal{S}'$, are those that were scheduled on M_i in the interval $\mathcal{I} \doteq [(k-B+1)^+, k+\ell')$ in $\mathcal{S}$. Note that the length of $\mathcal{I}$ is at most $\ell' + B - 1 \leq \ell$. For each job J_j, number its single units of processing scheduled on M_i in $\mathcal{I}$ in $\mathcal{S}$, as $U_{j,1}, U_{j,2}, \ldots$. Since the length of $\mathcal{I}$ is at most ℓ, the definition of C' shows that the number of such units for each job J_j, is at most $C'_{\mathcal{S},\ell}(i,(k-B+1)^+,j,j)$.

Let $X_{j,t}$ be the indicator random variable for $U_{j,t}$ getting scheduled in the interval $[k,k+\ell')$ in $\mathcal{S}'$. We have

$$C'_{\mathcal{S}',\ell'}(i,k,j_1,j_2) \leq \sum_{j=j_1}^{j_2} \sum_t X_{j,t}. \tag{4.1}$$

Since $\mathrm{E}[X_{j,t}] \leq \ell'/B$ for each j, t, we have

$$(4.2)\qquad \nu \doteq \mathrm{E}[C'_{\mathcal{S}',\ell'}(i,k,j_1,j_2)] \leq \sum_{j=j_1}^{j_2} (C'_{\mathcal{S},\ell}(i,(k-B+1)^+,j,j)\ell'/B) = C'_{\mathcal{S},\ell}(i,(k-B+1)^+,j_1,j_2)\ell'/B.$$

We now handle part (a). By (4.1), $C'' \doteq C'_{\mathcal{S}',\ell'}(i,k,j_1,j_2)/u$ is at most $\sum_{j=j_1}^{j_2} Y_j$, where $Y_j \doteq u^{-1}\sum_t X_{j,t}$. By the definition of u, $Y_j \leq 1$ for each j. So the random variables $\{Y_j : j \in [j_1, j_2]\}$ lie in $[0,1]$, and are independent. A Chernoff-Hoeffding bound shows for any $\delta' \geq 0$ that

$$(4.3)\quad \Pr[C'_{\mathcal{S}',\ell'}(i,k,j_1,j_2) \geq \nu(1+\delta')] = \Pr[C'' \geq (\nu/u)\cdot(1+\delta')] \leq G(\nu/u, \delta').$$

Next, $C'_{\mathcal{S},\ell}(i,(k-B+1)^+,j_1,j_2) \leq C$, since $\mathcal{S}$ is an (L,ℓ,C)-schedule. So, applying (4.2) and Fact 4.1(b) to (4.3) completes the proof for part (a).

For part (b), consider any job J_j. Since $\ell' = w$ here, the definition of w-separated shows that we cannot have $X_{j,t} = X_{j,t'} = 1$, if $t \neq t'$. This easily leads us to see that the random variables $\{X_{j,t} : j,t\}$ are negatively correlated, in the sense of Lemma 4.2. So, an application of Lemma 4.2 and Fact 4.1(b) to (4.1) and (4.2) completes the proof for part (b). □

We will next use these results to prove Theorem 1.3, in Sections 4.2 and 4.3.

4.2. Proof of Theorem 1.3(b). Recall that we are considering any w-separated job-shop instance I with $p_{\max} = 1$ now. Let $\mathcal{S}$ be the 0-delayed schedule for I. Thus, $\mathcal{S}$ is a $(P_{\max},\ell,C)$-schedule, where, e.g., $\ell = \Pi_{\max} + w - 1$ and $C = \Pi_{\max}$. Also let $\mathcal{S}'$ be the random $(B,\mathcal{S})$-delayed schedule, where $B = \Pi_{\max}$; i.e., $\mathcal{S}'$ is the random B-delayed schedule for I. Define $\ell' = w$.

We can ensure that $G(w,\delta) \leq (P_{\max} + \Pi_{\max})^{-c_3}$, by choosing (i) $\delta = c''$ if $w \geq \log(P_{\max}+\Pi_{\max})/2$, and (ii) $\delta = c''\log(P_{\max}+\Pi_{\max})/(w\log(\log(P_{\max}+\Pi_{\max})/w))$ if $w < \log(P_{\max}+\Pi_{\max})/2$, for some suitably large constant c''. By Lemma 4.5(b) and Theorem 4.3, we can then efficiently construct a $(P_{\max}+\Pi_{\max}, w, w(1+3\delta))$-schedule $\mathcal{S}''$ for I. We partition $\mathcal{S}''$ into $\lceil (P_{\max}+\Pi_{\max})/w \rceil$ intervals each of length w; crucially, each of these intervals (subproblems) is an *acyclic* job-shop instance. Also, in each of these subproblems, $p_{\max} = 1$, and any machine has at most $w(1+3\delta)$ operations to be scheduled on it. Via the result of [12], each subproblem can be efficiently scheduled with makespan $\mathrm{O}(w + w(1+\delta)) = \mathrm{O}(w(1+\delta))$. We then concatenate all these schedules, leading to a final makespan of $\mathrm{O}((P_{\max}+\Pi_{\max})(1+\delta))$.

4.3. Proof of Theorem 1.3(a). Recall that our goal is to show the existence of a schedule with makespan $\mathrm{O}((P_{\max}+\Pi_{\max})\cdot \frac{\log u}{\log\log u}\cdot \left\lceil \frac{\log(\min\{m\mu, p_{\max}\})}{\log\log u}\right\rceil)$, assuming that every job needs at most u time units on each machine. We assume that $u \geq 2$. Indeed, if $u = 1$, then we have an acyclic job-shop instance with $p_{\max} = 1$; so we will be able to efficiently construct a schedule of length $\mathrm{O}(P_{\max}+\Pi_{\max})$ [11, 12]. The algorithm is presented in Section 4.3.2; we start with a useful tool.

4.3.1. L'-splitting. Suppose we are given an (L,ℓ,C)-schedule $\mathcal{S}$ for a job-shop instance I, and want to split it into subproblems each of makespan at most L', where $p_{\max} < L' < L$. If $p_{\max} = 1$, this is easy, as seen in Section 4.2. Consider the case where $p_{\max}$ is arbitrary. We now show a simple way of partitioning the operations of $\mathcal{S}$ into at most $\lceil L/(L'-p_{\max})\rceil$ subproblems $\mathcal{P}_1, \mathcal{P}_2, \ldots$. We will also output an

(L', ℓ, C)-schedule $\mathcal{S}_i$ for each $\mathcal{P}_i$. These subproblems will be such that they can be solved independently and the resulting schedules concatenated to give a feasible schedule for I. This "L'-splitting" process is as follows.

We consider all operations that are completely finished by time L' in $\mathcal{S}$; scheduling this set of operations becomes our first subproblem $\mathcal{P}_1$. $\mathcal{S}$ provides a natural (L', ℓ, C)-schedule $\mathcal{S}_1$ for $\mathcal{P}_1$. If we have covered all operations by this process, we stop; if not, we define the next subproblem $\mathcal{P}_2$ as follows. Define $t_1 = 0$. Let t_2 be the smallest integer such that: (i) $t_2 \leq L'$, and (ii) there is some operation O starting at time t_2 in $\mathcal{S}$, such that O is not completely finished by time L'. (Note that $L' - p_{\max} < t_2 \leq L'$.) Our second subproblem $\mathcal{P}_2$ consists of all operations: (a) finishing by time $t_2 + L'$ in $\mathcal{S}$, and (b) not covered by $\mathcal{P}_1$. The time interval $[t_2, t_2 + L')$ in $\mathcal{S}$ provides an (L', ℓ, C)-schedule $\mathcal{S}_2$ for $\mathcal{P}_2$ in the obvious way. Once again, if we have not covered all operations, we define t_3 to be the smallest integer such that: (i) $t_3 \leq t_2 + L'$, and (ii) there is some operation O starting at time t_3 in $\mathcal{S}$, such that O is not completely finished by time $t_2 + L'$. We have $t_3 > t_2 + L' - p_{\max}$; thus $t_3 > 2(L' - p_{\max})$. $\mathcal{P}_3$ consists of all operations finishing by time $t_3 + L'$ that were not covered by $\mathcal{P}_1$ and $\mathcal{P}_2$. We iterate this until all operations are covered.

In general, we have $t_{i+1} \geq i(L' - p_{\max})$; so the total number of subproblems created is at most $\lceil L/(L' - p_{\max}) \rceil$. It is also easy to see that we have an (L', ℓ, C)-schedule $\mathcal{S}_i$ for each $\mathcal{P}_i$. Also, the subproblems can be solved independently and the resulting schedules concatenated to give a feasible schedule for I.

4.3.2. Algorithm and analysis. We choose a sufficiently large positive constant b_0. Define $L_0 = P_{\max} + \Pi_{\max}$, and $L_i = \log L_{i-1}$ for $i \geq 1$. We repeat this iteration until we arrive at a t for which either $L_{t+1} \geq L_t$, or $L_{t+1} \leq 36 b_0^2$. (Thus, the iteration proceeds for $\mathrm{O}(\log^*(P_{\max} + \Pi_{\max}))$ steps.) Also, for $1 \leq i \leq t$, define

$$C_i \doteq L_i^3 (1 + \frac{b_0}{\sqrt{L_1}}) \prod_{j=1}^{i-1} ((1 + \frac{b_0}{\sqrt{L_{j+1}}}) \cdot \frac{1}{1 - (L_{j+1}/L_j)^3}). \tag{4.4}$$

Recall that $L_i \geq 36 b_0^2$ for $1 \leq i \leq t$. If b_0 is large enough, we have

$$C_i \leq L_i^3 \exp\left((\sum_{j=1}^{i} \frac{b_0}{\sqrt{L_j}}) + \mathrm{O}(\sum_{j=1}^{i-1} (\frac{L_{j+1}}{L_j})^3) \right) \leq L_i^3 \exp(3 b_0/\sqrt{L_i}) \leq 2 L_i^3. \tag{4.5}$$

The second inequality follows from the fact that the terms L_j^{-1} increase exponentially, with $L_j^{-1} \leq (36 b_0^2)^{-1}$ and b_0 sufficiently large.

The algorithm is as follows. First, if $u^2 > P_{\max} + \Pi_{\max}$, then Corollary 4.4 shows that we can construct a schedule of makespan as claimed by Theorem 1.3(a). So suppose $u^2 \leq P_{\max} + \Pi_{\max}$. The algorithm consists of a preprocessing step and a general (recursive) step, motivated by the approach of Section 6.1 of [17].

Preprocessing step. We start with the obvious $(P_{\max}, \ell, \Pi_{\max})$-schedule $\mathcal{S}$, where ℓ can be taken arbitrarily large.

We first handle the case where $u \geq b_0 L_1$. We call this the "simple case". Define $\ell' = u^2$, $B = \Pi_{\max}$, and $\delta = 1$. If b_0 is large enough, then $G(u, \delta) \leq (P_{\max} + \Pi_{\max})^{-c_3}$. Thus, by Lemma 4.5(a) and Theorem 4.3, we can efficiently construct a $(P_{\max} + \Pi_{\max}, u^2, 4u^2)$-schedule $\mathcal{S}'$. We apply u^2-splitting to $\mathcal{S}'$, as defined in Section 4.3.1. Since $u \geq 2$ and $p_{\max} \leq u$, the total number of subproblems is at most $\lceil (P_{\max} + \Pi_{\max})/(u^2 - p_{\max}) \rceil \leq \mathrm{O}((P_{\max} + \Pi_{\max})/u^2)$. Also, each of the subproblems has "$P_{\max}$" at most u^2 and "$\Pi_{\max}$" at most $4u^2$. So, by Corollary 4.4, each of these subproblems can be efficiently given a valid schedule of makespan $\mathrm{O}\left(u^2 \cdot \frac{\log u}{\log\log u} \cdot \left\lceil \frac{\log(\min\{m\mu, p_{\max}\})}{\log\log u} \right\rceil\right)$. As seen above, the number of subproblems is $\mathrm{O}((P_{\max} + \Pi_{\max})/u^2)$, so the concatenation of these schedules yields a final schedule of makespan as claimed by Theorem 1.3(a).

We now move on to the more interesting case where $u < b_0 L_1$. We define $\ell' = L_1^3$, $B = \Pi_{\max}$, and $\delta = b_0/(3\sqrt{L_1})$. By Fact 4.1(a) and since $u < b_0 L_1$, we have $G(\ell'/u, \delta) \leq \exp(-b_0 L_1/27)$, which can be made at most $(P_{\max} + \Pi_{\max})^{-c_3}$ if b_0 is chosen sufficiently large. By Lemma 4.5(a) and Theorem 4.3, we can efficiently construct a $(P_{\max} + \Pi_{\max}, L_1^3, C_1)$-schedule $\mathcal{S}'$. (See (4.4) for the definition of the C_i.) We apply L_1^4-splitting to $\mathcal{S}'$ to obtain some subproblems, each of which also comes with an (L_1^4, L_1^3, C_1)-schedule. The number of subproblems is at most

$$\begin{aligned}
\lceil L_0/(L_1^4 - p_{\max}) \rceil &\leq \lceil L_0/(L_1^4 - b_0 L_1) \rceil \\
&\leq L_0/(L_1^4 - b_0 L_1) + 1 \\
&\leq \frac{L_0}{L_1^4} \cdot (1 + \mathrm{O}(1/L_1^3) + \mathrm{O}(L_1^4/L_0)) \\
&\leq \frac{L_0}{L_1^4} \cdot (1 + \mathrm{O}(1/L_1^3)). \qquad (4.6)
\end{aligned}$$

We next show a recursive scheme to handle each of these subproblems.

General step. Suppose, in general, we have a subproblem which comes with an (L_i^4, L_i^3, C_i)-schedule, $1 \leq i \leq t$. We first dispose of some easy cases. If $i = t$, then $L_i = \mathrm{O}(1)$; by (4.5), $C_i = \mathrm{O}(1)$ also. Thus, we can efficiently find a schedule of length $\mathrm{O}(1)$. So we assume $i \leq t - 1$. Next, suppose $u^2 \geq L_i^3/2$. Note that the "$P_{\max}$" and "$\Pi_{\max}$" values of the given subproblem are respectively at most L_i^4 and $C_i \cdot (L_i^4/L_i^3) = \mathrm{O}(L_i^4)$. Thus, if $u^2 \geq L_i^3/2$, then Corollary 4.4 shows that we can construct a schedule of makespan

$$\mathrm{O}\left(L_i^4 \cdot \frac{\log u}{\log\log u} \cdot \left\lceil \frac{\log(\min\{m\mu, p_{\max}\})}{\log\log u} \right\rceil\right). \qquad (4.7)$$

So we assume that $u^2 < L_i^3/2$.

We now show a scheme that will construct a feasible schedule for the problem if $u \geq b_0 L_{i+1}$; if $u < b_0 L_{i+1}$, we will show how to reduce this problem to a number of subproblems, each of which comes with an $(L_{i+1}^4, L_{i+1}^3, C_{i+1})$-schedule.

First suppose $u \geq b_0 L_{i+1}$. We follow our approach for the simple case of the preprocessing step. Define $B = L_i^3/2$, $\ell = L_i^3$, $\ell' = u^2$, and $\delta = 1$. Since $u^2 < L_i^3/2$, we have $B + \ell' \leq \ell$ as required by Theorem 4.3. So, if b_0 is sufficiently large, we will have

$$G(C_i \ell'/(Bu), \delta) \leq (L_i^4 + C_i)^{-c_3}, \qquad (4.8)$$

since $L_i^3 < C_i \leq 2L_i^3$ by (4.4) and (4.5). As in the "simple case", we can get an $(L_i^4 + B, \ell', \mathrm{O}(\ell'))$-schedule, apply ℓ'-splitting to it, and solve the resulting subproblems using Corollary 4.4. The final schedule will have makespan as in (4.7).

Finally, suppose $u < b_0 L_{i+1}$. We follow the general idea of the "interesting case" of the preprocessing step. Define $B = L_i^3 - L_{i+1}^3$, $\ell = L_i^3$, $\ell' = L_{i+1}^3$, and $\delta = b_0/(3\sqrt{L_{i+1}})$. Once again, since $u < b_0 L_{i+1}$, we will have (4.8). Thus, as in the "interesting case", we construct an $(L_i^4 + L_i^3, L_{i+1}^3, C_{i+1})$-schedule, and apply L_{i+1}^4-splitting to it. As a result, we get some number of subproblems, each of which is equipped with an $(L^4_{i+1}, L^3_{i+1}, C_{i+1})$-schedule; we recurse on these independently. As in the derivation of (4.6), the number of subproblems is at most

$$\lceil (L_i^4 + L_i^3)/(L_{i+1}^4 - b_0 L_{i+1}) \rceil \leq \frac{L_i^4}{L_{i+1}^4} \cdot (1 + \mathrm{O}(1/L_{i+1}^3)). \tag{4.9}$$

Let the final set of subproblems we solve be those that come with an (L_p^4, L_p^3, C_p)-schedule, for some p. The product of the terms in (4.6) and (4.9) as i runs from 1 to $p-1$, is $\mathrm{O}(L_0/L_p^4)$. Thus, by (4.7), the final makespan is

$$\begin{aligned} &\mathrm{O}\left((L_0/L_p^4)L_p^4 \cdot \frac{\log u}{\log\log u} \cdot \left\lceil \frac{\log(\min\{m\mu, p_{\max}\})}{\log\log u} \right\rceil\right) \\ &\qquad = \mathrm{O}\left(L_0 \cdot \frac{\log u}{\log\log u} \cdot \left\lceil \frac{\log(\min\{m\mu, p_{\max}\})}{\log\log u} \right\rceil\right), \end{aligned}$$

as claimed by Theorem 1.3(a).

4.4. Basic ideas from earlier constructivizations of the LLL. This section is based on the work of [5, 1, 12]. The main result here is Theorem 4.7, which will be used in Section 4.5 to prove Theorem 4.3.

Given an undirected graph $G = (V, E)$, recall that a set $C \subseteq V$ is a *dominating set* of G if and only if all vertices in $V - C$ have some neighbour in C. For any positive integer ℓ, we define G^ℓ to be the graph on the same vertex set V, with two vertices adjacent if and only if they are distinct **and** there is a path of length at most ℓ that connects them in G. We let $\Delta(G)$ denote the maximum degree of the vertices in G. Also, suppose $\mathcal{R}$ is some random process and that each vertex in V represents some event related to $\mathcal{R}$. We say that G is a *dependency graph for* $\mathcal{R}$ if and only if for each $v \in V$ and any set of vertices S such that no element of S is adjacent to v in G, we have that the event corresponding to v is independent of any Boolean combination of the events corresponding to the elements of S.

In Lemma 4.6 and subsequently, the phrase "connected component" means "*maximal* connected subgraph", as usual.

LEMMA 4.6. *Given an undirected graph $G_1 = (V, E)$ with a dominating set C, let G_2 be the subgraph of G_1^3 that is induced by C. Pick an arbitrary maximal independent set I in G_2, and let G_3 be the subgraph of G_2^3 induced by I. Suppose G_1 has a connected component with N vertices. Then G_3 has a connected component with at least $N/((\Delta(G_1)+1)(\Delta(G_1))^3)$ vertices.*

Proof. Let $C_1 = (U, E')$ be a connected component of G_1 with N vertices. Then, the vertices in $C \cap U$ are connected in G_2, which is seen as follows. Suppose $v_1, u_1, u_2, \ldots, u_t, v_t$ is a path in C_1, where v_1 and v_t are in $C \cap U$, and $u_1, \ldots, u_t$ are all in $U - (C \cap U)$. Then, since $C \cap U$ is a dominating set in C_1, for $1 < i < t$, u_i must have some neighbour $v_i \in C \cap U$. Hence, there are paths $v_i, u_i, u_{i+1}, v_{i+1}$ for $1 \leq i < t$ so v_1 and v_t are connected in G_2. Thus, all of the vertices in $C \cap U$ are connected in G_2. Since $C \cap U$ is a dominating set in C_1, it is also easy to check that $|C \cap U| \geq N/(\Delta(C_1)+1) \geq N/(\Delta(G_1)+1)$. Thus, $C \cap U$ yields a connected component C_2 in G_2 that has at least $N/(\Delta(G_1)+1)$ vertices.

Since $\Delta(C_2) \leq (\Delta(G_1))^3 - 1$, one can similarly show that $I \cap (C \cap U)$ yields a connected component C_3 in G_3 that has at least

$$\frac{|C \cap U|}{\Delta(C_2)+1} \geq \frac{|C \cap U|}{(\Delta(G_1))^3} \geq \frac{N}{(\Delta(G_1)+1)(\Delta(G_1))^3}$$

vertices. □

We present a key ingredient of [5, 1, 12]:

THEOREM 4.7. *Let a graph $G = (V, E)$ be a dependency graph for a random process $\mathcal{R}$, with the probability of occurrence of the event represented by any vertex of G being at most r. Run the process $\mathcal{R}$, and let $C \subseteq V$ be the vertices of G that represent the events (among the elements of V) that occurred during the run. (Thus, C is a random subset of V with some distribution.) Let G_1 be the subgraph of G induced by $C \cup C'$, where C' is the set of vertices of G that have at least one neighbour in C. Then, for any $x \geq 1$, the probability of G_1 having a connected component with at least $x(\Delta(G)+1)(\Delta(G))^3$ vertices, is at most $|V|\Delta(G)^{-18}\sum_{y \geq x}(\Delta(G)^{18}r)^y$.*

Proof. Observe that, by construction, C is a dominating set for G_1. Construct G_2, I, and G_3 as in the statement of Lemma 4.6. Note that deterministically, $\Delta(G_1) \leq \Delta(G)$. Thus, by Lemma 4.6, we just need to bound the probability of G_3 having a connected component with x or more vertices.

Suppose that a size-y set S of vertices of G forms a connected component in G_3. Then there is a sub-tree T of G_3 which spans the vertices in S. T can be represented by a list L which lists all of the vertices that are visited in a depth-first traversal of T. Each vertex in T (except the root) is visited both before its children and after each child (the root is only visited after each child), so each vertex appears on L once for each edge adjacent to it in T. Thus, the length of L is $2(y-1)$. If two vertices are adjacent on L then they are adjacent in G_3, which implies that the distance between them in G is at most 9. Thus, given G, the number of possible sets S is at most the number of possible lists L, which is at most $|V|$ (the number of choices for the first vertex on L) times $(\Delta(G)^9)^{2(y-1)}$ (the number of choices for the rest of L). Thus, the number of sets S which could possibly correspond to size-y connected components in G_3 is at most $|V|\Delta(G)^{-18}\Delta(G)^{18y}$.

The definition of I implies that the vertices in G_3 form an independent set in G. Furthermore, given any independent set S of size y in G, Bayes' theorem and the definition of dependency graphs show that the probability that all elements of S are in G_3 is at most r^y. Thus, the probability that G_3 has a connected component of size y is at most $|V|\Delta(G)^{-18}(\Delta(G)^{18}r)^y$. □

4.5. Proof of Theorem 4.3. We now assume the notation of Theorem 4.3 and prove the theorem. Define the following "bad" events:

$$\mathcal{E}(i,k,j_1,j_2) \equiv (C'_{\mathcal{S}',\ell'}(i,k,j_1,j_2) \geq \frac{\ell'}{B} \cdot (C'_{\mathcal{S},\ell}(i,(k-B+1)^+,j_1,j_2) + C\delta));$$
$$\mathcal{E}'(i,k,j_1) \equiv (\exists j_2 \geq j_1 : \ \mathcal{E}(i,k,j_1,j_2)).$$

By the assumption of Theorem 4.3, $\Pr[\mathcal{E}(i,k,j_1,j_2)] \leq (\max\{L,B,C\})^{-c_3}$ for all (i,k,j_1,j_2). Now, for the given instance I, $P_{\max} \leq L$ and $\Pi_{\max} \leq C \cdot \lceil L/\ell \rceil \leq CL$. Thus, in particular, at most CL jobs use any given machine M_i. So, we have for all (i,k,j_1) that

$$\Pr[\mathcal{E}'(i,k,j_1)] \leq p \doteq CL(\max\{L,B,C\})^{-c_3}. \tag{4.10}$$

The algorithm processes the jobs in the order $J_1, J_2, \ldots$. When it is job J_j's turn, we give it a random delay from $\{0, 1, \ldots, B-1\}$, and check if this makes, for any pair (i,k), the event $\mathcal{E}(i,k,1,j)$ true. If so, we temporarily set aside J_j and all *yet-unprocessed* jobs that use machine M_i. Let $\mathcal{J}_1$ denote the set of jobs which **do** get assigned a delay by this process. We shall basically show that, with high probability, the problem of assigning delays to the jobs not in $\mathcal{J}_1$ gets decomposed into a set of much smaller subproblems. To this end, we first set up some notation in order to apply Theorem 4.7.

Construct an undirected graph G with the events $\mathcal{E}'(i,k,1)$ as nodes, with an edge between two distinct nodes $\mathcal{E}'(i,k,1)$ and $\mathcal{E}'(i',k',1)$ if and only if either (P1) $i=i'$, or (P2) there is some job that uses both the machines M_i and $M_{i'}$. It is easy to check that G is a valid dependency graph for the events $\mathcal{E}'(i,k,1)$. The number of vertices in G is at most $m(L+B)$. Recall that at most CL jobs use any given machine and that each such job uses at most $L-1$ other machines. Thus, each node can have at most $L+B$ neighbours of type (P1), and at most $CL(L-1)(L+B)$ neighbours of type (P2). So $\Delta(G)$ is at most

$$L+B+CL(L-1)(L+B) \leq CL^2(L+B)-1.$$

Run the above random process of randomly scheduling and setting aside (if necessary) some of the jobs. Let $\mathcal{C}$ be the set of events $\mathcal{E}'(i,k,1)$ that actually happened. Let $\mathcal{C}'$ be the set of nodes of G that have at least one neighbour in $\mathcal{C}$, and let G_1 be the subgraph of G that is induced by $\mathcal{C}\cup\mathcal{C}'$. Thus, by applying Theorem 4.7 with $|V|\leq m(L+B)$, $\Delta(G)\leq CL^2(L+B)-1$, $x=\log m$ and $r=p$, we see from (4.10) that

$$\begin{aligned}&\Pr[G_1 \text{ has a connected component with at least } (CL^2(L+B))^4\log m \text{ nodes}]\\ &\leq 1/2, \qquad (4.11)\end{aligned}$$

if c_3 is appropriately large.

We repeat the above process until all connected components of G_1 have at most $(CL^2(L+B))^4\log m$ nodes. By (4.11), we expect to run the above process at most twice.

What have we achieved? Let us first give all the jobs in $\mathcal{J}_1$ their assigned delays, and remove them from consideration. The key observation is as follows. Fix any remaining job J_j. Then, for **no two** machines M_i and $M_{i'}$ that are both used by J_j, can we have two nodes $\mathcal{E}'(i,k,1)$ and $\mathcal{E}'(i',k',1)$ in *different* connected components of G_1. This is because $\mathcal{E}'(i,k,1)$ and $\mathcal{E}'(i',k',1)$ are neighbours in G. Thus, the problem in each connected component of G_1 can be solved completely *independently* of the other connected components.

So all connected components of G_1 have at most $(CL^2(L+B))^4\log m$ nodes. To further reduce this component size, we repeat the above process on each connected component CC_t of G_1 separately, as follows. Fix any such CC_t. Define $f_1(i)$ to be the least index j such that $J_j\notin\mathcal{J}_1$ and such that J_j uses M_i. (If all jobs that use M_i are in $\mathcal{J}_1$, we define $f_1(i)=n+1$ for convenience.) Note that all jobs J_j that use M_i and have $j\geq f_1(i)$, lie outside the set $\mathcal{J}_1$. We process the jobs lying outside $\mathcal{J}_1$ in order as before. When it is job J_j's turn, we give it a random delay from $\{0,1,\ldots,B-1\}$, and check if this makes, for any pair (i,k), the event $\mathcal{E}(i,k,f_1(i),j)$ true. (This is mostly the same as before, except that we have "$f_1(i)$" in place of "1" now.) If so, we temporarily set aside J_j and all yet-unprocessed jobs lying outside $\mathcal{J}_1$, that use M_i.

We proceed similarly as above. Let $\mathcal{J}_2$ denote the set of jobs which get assigned a delay by this process. We now show that the problem of assigning delays to the jobs not in $\mathcal{J}_1\cup\mathcal{J}_2$ gets decomposed into even smaller subproblems, with high probability. In place of the bad events $\{\mathcal{E}'(i,k,1)\}$, the bad events now are $\{\mathcal{E}'(i,k,f_1(i))\}$. We can once again invoke Theorem 4.7; we take $|V|\leq(CL^2(L+B))^4\log m$, $\Delta(G)\leq$

$CL^2(L+B)-1$, $x = \log\log m$ and $r = p$. As before, if c_3 is large enough, we expect to repeat this process at most twice before ensuring that all resulting "connected components" have at most $(CL^2(L+B))^4 \log\log m$ nodes.

We now consider any connected component CC'_t remaining after the above two passes. (Once again, all these components can be handled independently.) Define $f_2(i)$ to be the least index j such that $J_j \notin (\mathcal{J}_1 \cup \mathcal{J}_2)$ and such that J_j uses M_i. We now show how to give delays to all jobs lying outside $(\mathcal{J}_1 \cup \mathcal{J}_2)$, in a manner that avoids all the events $\mathcal{E}'(i, k, f_2(i))$. There are two cases:

Case I: $\log\log m \leq L + B + C$. In this case, the number of "nodes" (events $\mathcal{E}'(i, k, f_2(i))$) in CC'_t is $\mathrm{poly}(L, B, C)$. Thus, if we start with a random B-delayed schedule for the jobs associated with CC'_t, the probability that at least one "bad" event associated with CC'_t (i.e., at least one node of CC'_t) happens is at most $1/2$, if c_3 is large enough. So we expect to run this process on CC'_t at most twice.

Case II: $\log\log m > L+B+C$. The number of nodes in CC'_t is $\mathrm{O}(\mathrm{poly}(\log\log m))$ in this case. So the number of machines associated with CC'_t is also $\mathrm{O}(\mathrm{poly}(\log\log m))$, and hence the number of jobs associated with CC'_t is at most $\mathrm{O}(L \cdot \mathrm{poly}(\log\log m))$, i.e., $\mathrm{O}(\mathrm{poly}(\log\log m))$.

We recall the Lovász Local Lemma (LLL):

LEMMA 4.8. **([7])** *Let $E_1, E_2, \ldots, E_\ell$ be any events with $\Pr[E_i] \leq q$ for all i. If each E_i is mutually independent of all but at most d of the other events E_j and if $eq(d+1) \leq 1$, then $\Pr[\bigwedge_{i=1}^{\ell} \overline{E_i}] > 0$.*

As seen above, any event $\mathcal{E}'(i, k, f_2(i))$ depends on at most $CL^2(L+B)-1$ other such events. Also, $\Pr[\mathcal{E}'(i, k, f_2(i))] \leq p$ for all i, k. Thus, if c_3 is sufficiently large, the LLL shows that *there exists* a way of giving a delay in $\{0, 1, \ldots, B-1\}$ to each job associated with CC'_t, in order to avoid all the events $\mathcal{E}'(i, k, f_2(i))$ associated with CC'_t. But here, there are at most $\mathrm{O}(\mathrm{poly}(\log\log m))$ jobs, and each has only $B \leq \log\log m$ possible initial delays! Thus, *exhaustive search* can be applied to find a "good" B-delayed schedule that we know to exist: the time needed for CC'_t is at most

$$(\log\log m)^{\mathrm{O}(\mathrm{poly}(\log\log m))} = m^{o(1)}.$$

Let $\mathcal{S}''$ be the final delayed schedule produced. Consider any interval $(k, k+\ell')$. We have

$$\begin{aligned}
C_{\mathcal{S}'',\ell'}(i,k) &\leq \frac{\ell'}{B} \cdot (C'_{\mathcal{S},\ell}(i, (k-B+1)^+, 1, f_1(i)-1) + C\delta) + \\
&\quad \frac{\ell'}{B} \cdot (C'_{\mathcal{S},\ell}(i, (k-B+1)^+, f_1(i), f_2(i)-1) + C\delta) + \\
&\quad \frac{\ell'}{B} \cdot (C'_{\mathcal{S},\ell}(i, (k-B+1)^+, f_2(i), n) + C\delta) \\
&= \frac{\ell'}{B} \cdot (C'_{\mathcal{S},\ell}(i, (k-B+1)^+, 1, n) + 3C\delta) \\
&\leq \frac{\ell'}{B} \cdot (C + 3C\delta),
\end{aligned}$$

as required.

It is also easy to check via linearity of expectation that the expected running time of the algorithm is $\mathrm{poly}(m, L, \ell, C)$. This concludes the proof of Theorem 4.3.

Acknowledgements. Aravind Srinivasan thanks David Shmoys for introducing him to this area, for sharing many of his insights, and for his several suggestions. He also thanks Cliff Stein and Joel Wein for their suggestions and many helpful discussions. We thank Uri Feige for bringing the work of [16] and [22] to our attention, and Bruce Maggs and Andréa Richa for sending us an updated version of [12]. We thank the referees for their many helpful suggestions.

REFERENCES

[1] N. Alon, *A parallel algorithmic version of the Local Lemma*, Random Structures & Algorithms, 2 (1991), pp. 367–378.

[2] N. Alon and A. Srinivasan, *Improved parallel approximation of a class of integer programming problems*, Algorithmica, 17 (1997), pp. 449–462.

[3] D. Applegate and W. Cook, *A computational study of the job-shop scheduling problem*, ORSA Journal of Computing, 3 (1991), pp. 149–156.

[4] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber, *Bandwidth allocation with preemption*, in Proc. ACM Symposium on Theory of Computing, 1995, pp. 616–625.

[5] J. Beck, *An algorithmic approach to the Lovász Local Lemma*, Random Structures & Algorithms, 2 (1991), pp. 343–365.

[6] J. Carlier and E. Pinson, *An algorithm for solving the job-shop problem*, Management Science, 35 (1989), pp. 164–176.

[7] P. Erdős and L. Lovász, *Problems and results on 3-chromatic hypergraphs and some related questions*, in Infinite and Finite Sets, A. Hajnal et. al., editors, Colloq. Math. Soc. J. Bolyai 11, North Holland, Amsterdam, 1975, pp. 609–627.

[8] U. Feige and C. Scheideler, *Improved bounds for acyclic job shop scheduling*, in Proc. ACM Symposium on Theory of Computing, 1998, pp. 624–633.

[9] L. A. Hall, *Approximability of flow shop scheduling*, in Proc. IEEE Symposium on Foundations of Computer Science, 1995, pp. 82–91.

[10] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *Sequencing and scheduling: algorithms and complexity*, in Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory, S. C. Graves et al., editors, Elsevier, 1993, pp. 445–522.

[11] F. T. Leighton, B. Maggs, and S. Rao, *Packet routing and jobshop scheduling in* O(*congestion + dilation*) *steps*, Combinatorica, 14 (1994), pp. 167–186.

[12] F. T. Leighton, B. Maggs, and A. Richa, *Fast algorithms for finding* O(*congestion + dilation*) *packet routing schedules*. Combinatorica, 19 (1999), pp. 375–401.

[13] S. Mahajan, E. A. Ramos, and K. V. Subrahmanyam, *Solving some discrepancy problems in NC*, in Proc. Annual Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 1346, Springer, 1997, pp. 22–36.

[14] P. Martin and D. B. Shmoys, *A new approach to computing optimal schedules for the job-shop scheduling problem*, in Proc. MPS Conference on Integer Programming and Combinatorial Optimization, 1996, pp. 389–403.

[15] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[16] G. Rayzman, *Approximation techniques for job-shop scheduling problems,* MSc Thesis, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel, July, 1996.

[17] C. Scheideler, *Universal Routing Strategies for Interconnection Networks*, Lecture Notes in Computer Science 1390, Springer, 1998.

[18] J. P. Schmidt, A. Siegel, and A. Srinivasan, *Chernoff-Hoeffding bounds for applications with limited independence*, SIAM J. Discrete Math., 8 (1995), pp. 223–250.

[19] S. V. Sevast'yanov, *Efficient construction of schedules close to optimal for the cases of arbitrary and alternative routes of parts*, Soviet Math. Dokl., 29 (1984), pp. 447–450.

[20] S. V. Sevast'yanov, *Bounding algorithm for the routing problem with arbitrary paths and alternative servers*, Kibernetika, 22 (1986), pp. 74–79 (translation in Cybernetics 22, pp. 773–780).

[21] D. B. Shmoys, C. Stein, and J. Wein, *Improved approximation algorithms for shop scheduling problems*, SIAM J. Comput., 23 (1994), pp. 617–632.

[22] Yu. N. Sotskov and N. V. Shaklevich, NP*-hardness of scheduling problems with three jobs,* Vesti Akad. Navuk BSSR Ser. Fiz.-Mat. Navuk, 4 (1990), pp. 96–101 (in Russian).
[23] D. P. Williamson, L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevast'yanov, and D. B. Shmoys, *Short shop schedules*, Operations Research, 45 (1997), pp. 288–294.

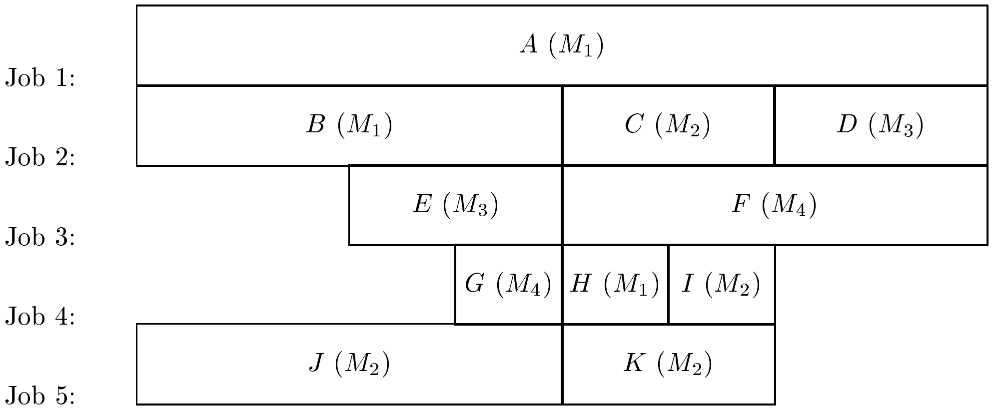

FIG. 3.1. *One frame of S, where $p_{\max} = 8$, A–K are the labels of operations, and M_1–M_4 are the machines.*

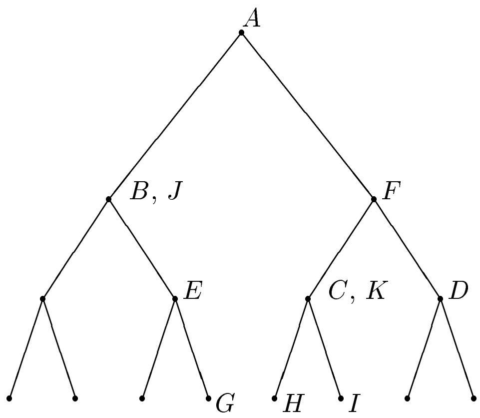

FIG. 3.2. *Assigning operations to nodes of T. For example, if u denotes the leftmost node on the second-highest level, then $S_1(u) = \{B\}$, $S_5(u) = \{J\}$, and $S_\ell(u) = \emptyset$ for every other ℓ.*

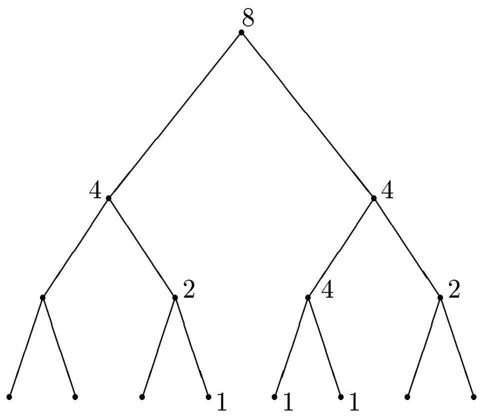

FIG. 3.3. *Calculating p for each node.*

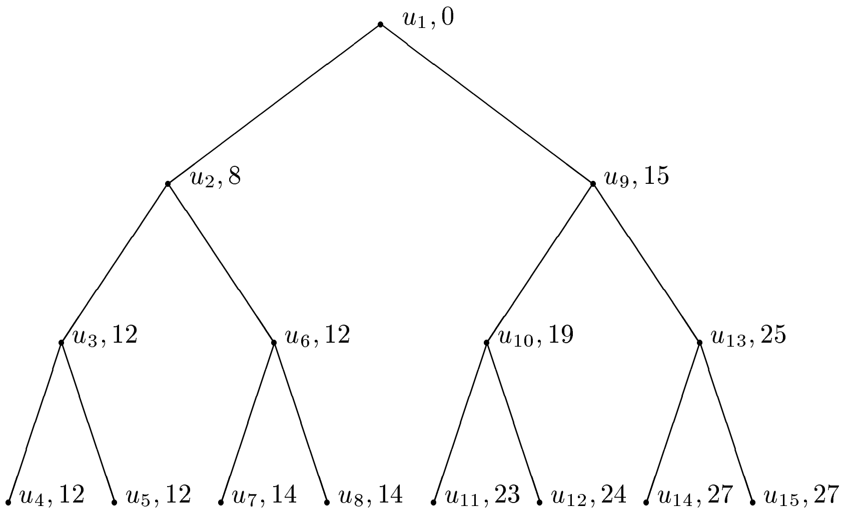

FIG. 3.4. *Calculating f for each node.*

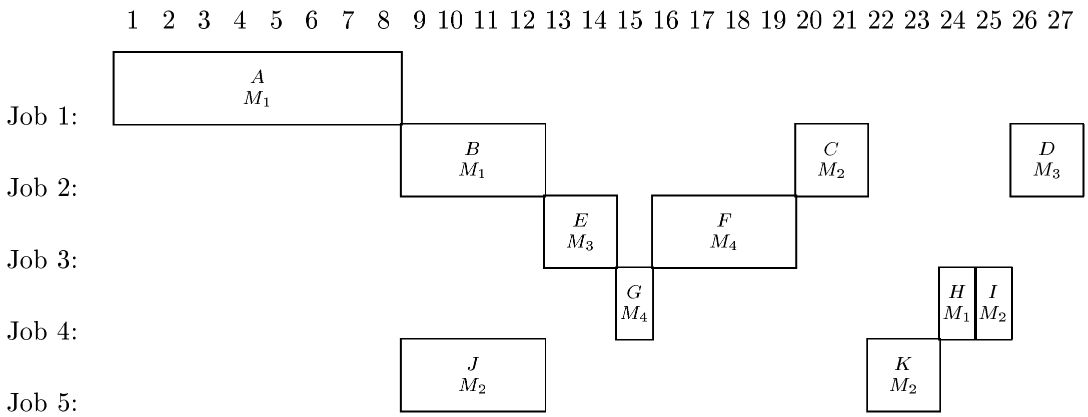

FIG. 3.5. *The schedule S'.*

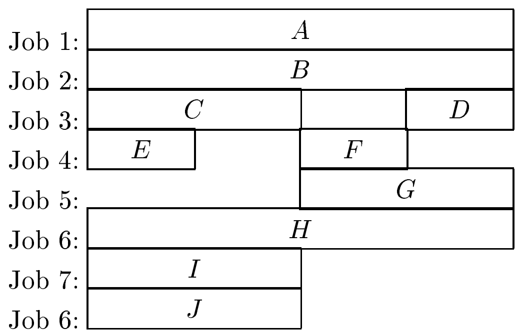

FIG. 3.6. *One frame of S, focussing* **only** *on operations for a single machine.*

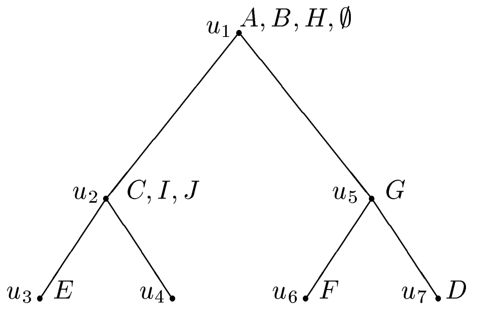

FIG. 3.7. *Assigning a dummy operation $\emptyset$ to the root of T'.*

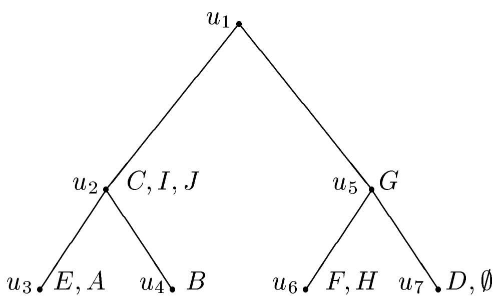

FIG. 3.8. *Redistributing the operations originally allocated to u_1.*

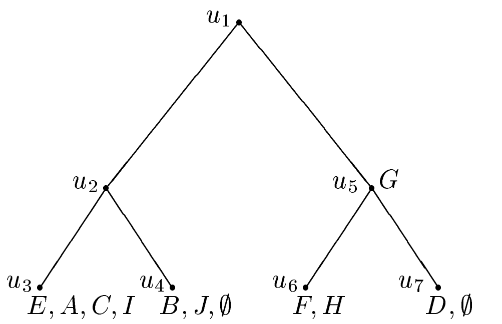

FIG. 3.9. *Redistributing the operations originally allocated to u_2.*

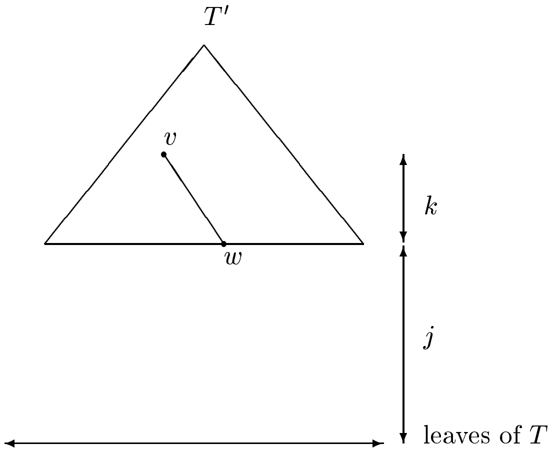

FIG. 3.10. *The case in which w is a leaf.*

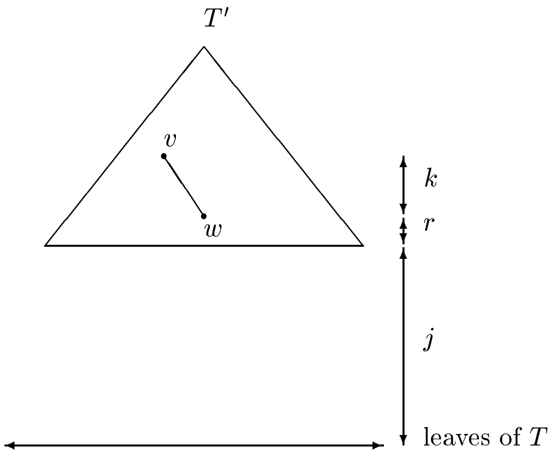

FIG. 3.11. *The case in which w is not a leaf.*