Cass Business School
CITY UNIVERSITY LONDON

Faculty of Actuarial Science and Insurance

Statistical Research Paper
No. 30

Scheduling Reentrant Jobs on Parallel Machines with a Remote Server

Konstantin Chakhlevitch
Celia Glass

October 2008

ISBN 978-1-905752-18-8

Cass Business School
106 Bunhill Row
London EC1Y 8TZ
Tel +44 (0)20 7040 8470
www.cass.city.ac.uk

Scheduling reentrant jobs on parallel machines with a remote server

K. Chakhlevitch* and C.A. Glass
Cass Business School, City University London, 106 Bunhill Row, EC1Y 8TZ, UK
Konstantin.Chakhlevitch.1@city.ac.uk, C.A.Glass@city.ac.uk

Abstract

This paper explores a specific combinatorial problem relating to re-entrant jobs on parallel primary machines, with a remote server machine. A middle operation is required by each job on the server before it returns to its primary processing machine. The problem is inspired by the logistics of a semi-automated micro-biology laboratory. The testing programme in the laboratory corresponds roughly to a hybrid flowshop, whose bottleneck stage is the subject of study. We demonstrate the NP-hard nature of the problem, and provide various structural features. A heuristic is developed and tested on randomly generated benchmark data. Results indicate solutions reliably within 1.5% of optimum. We also provide a greedy 2-approximation algorithm. Test on real-life data from the microbiology laboratory indicate a 20% saving relative to current practice, which is more than can be achieved currently with 3 instead of 2 people staffing the primary machines.
Keywords: scheduling, parallel identical machines, remote server, reentrant jobs

1 Introduction

The scheduling problem considered in this paper arises from the technological process observed in a microbiological laboratory. The laboratory performs microbiological testing on samples of processed foods prior to release to retailers. Each food sample undergoes a suite of tests, which is specified in advance. Each test suite consists of a number of microbiological tests, carried out by scientists with the aid of semi-automated equipment. The samples proceed through a number of consecutive processes, which include weighing, pipetting out onto test plates and a period of incubation. According to their test suite, samples require different dilutions, and tests on the samples require different types of media and incubation times for bacterial development, among other things. The laboratory is equipped with a fully integrated Laboratory Information Management System (LIMS) which controls individual tests for each sample. Pathogen tests are carried out in a separate laboratory. Thus, all samples in the main laboratory may be processed without risk of contamination. The logistics of testing in the microbiology laboratory may be viewed as a type of hybrid flowshop, with sets of parallel machines, performing similar functions, at various stages. However, one of the stages, namely the pipetting stage, appears as the bottleneck stage in

*Corresponding author

the whole testing process. The test samples are pipetted onto test plates which must be assembled and labelled ready for the purpose. The test plates are prepared by a dedicated machine, which serves several pipetting stations. The interaction between the single plate labelling machine and the pipetters working in parallel, gives rise to an unusual scheduling problem which is the subject of this paper.

The problem may be formulated in terms of a set of m parallel primary machines, $M_1, M_2, M_3, \ldots, M_m$ (the pipetters) and a single server machine S (the plate labelling machine). Jobs have three operations, the first and last of which are performed by the same pipetter, M_i say, with a middle operation on the server, S. The issue is not one of sequencing operations, as the order of jobs and of each of their operations is predetermined in our context. The challenge is that of assigning jobs to the primary machines. We restrict attention to the case of identical machines. We refer to this as the Initialization-Setup-Processing, ISP, problem. To the best of our knowledge, the ISP problem has not been studied before.

The scenario described above can be modelled using a parallel machines with a common server environment. A range of problems thus classified has been considered in the literature. Complexity results for problems with identical parallel machines with a single server and different objective functions were obtained by Hall et al. [1], Brucker et al. [2] and Kravchenko and Werner [3]. Abdekhodaee and Wirth [4] studied some solvable cases of the makespan minimization problem. Several heuristics for a general case were developed by Abdekhodaee et al. in [5]. Glass et al. [6] analyzed scheduling problems in the environment with parallel dedicated machines and a single server. Note that all these papers deal with the problems where a server visits the machine to perform a setup operation for each job, thus making the machine unavailable for processing any job until the setup operation is completed. In our scenario, the setup operations for jobs are performed on a *remote* or *external server*, i.e. no machine is tied up by setup operation of a particular job and each machine is available at any time for processing another job for which setup has been completed earlier. Similar problems are briefly discussed in [7]; however, no solution method is presented.

An important distinguishing feature of our model is the presence of initialization operation for each job, prior to its setup operation, which leads to different rules for assigning jobs to the machines. This feature makes our problem relevant to the class of reentrant shop scheduling problems namely *chain-reentrant shop* problems studied by Wang et al. in [8]. In a chain-reentrant shop with m machines, each job is first processed on the primary machine M_1, then on secondary machines in the order $M_2, M_3, ..., M_m$ and returns to M_1 for its final operation. Finding a sequence of jobs which minimizes makespan in a chain-reentrant shop is shown to be NP-hard problem even for $m = 2$ (see [8]).

This paper studies the ISP problem. We establish its complexity, as NP-hard, and its relationship with the reentrant chain problem. The essence of the ISP problem is contained in the case of 2 primary machines, which is the subject of the remainder of the paper. We offer bounds and an heuristic algorithm for this case. The effectiveness of our approach is demonstrated in two different ways. We present results for the real-world laboratory data, which show a 20% potential improvement from using our heuristic. In addition, systematic tests on randomly generated data, illustrate the impact of the heuristic on a broader range of instances. Solutions are shown to be always within 2% of optimum. However, the relative impact, compared with the default scheduling strategy, of our heuristic approach varies with the nature of the problem instance. Further insights are provided by analysis of the computational results.

The rest of the paper is organized as follows. In Section 2 we describe the problem in more detail. The specific context of pipetting in the micro-biology laboratory is described more fully, and the derivation of the consequent mathematical formulation given. In Section 3, we prove that the ISP problem is NP-hard in the ordinary sense, and then consider various structural properties of an optimal solution. In particular, we relate the ISP problem more precisely to the chain re-entrant problem in the literature. We also explore the impact of the various restrictions of this very specific problem to put it in a broader context. In section 4 we present our proposed heuristic algorithm. We also describe a branch and bound algorithm which we use to compute an optimal solutions and a default scheduling algorithm currently used in the laboratory at the pipetting stage. These algorithms are used for performance assessment of our proposed heuristic. Computational experiments are reported and analyzed in section 5 where we study the application of our heuristic to the real-world version of the problem as well as analyse the performance of the heuristic for the broad range of randomly generated data. Finally, Section 6 contains some concluding remarks.

2 Problem Description

2.1 Microbiology laboratory context

The re-sequencing of the samples is possible only in advance or at the earliest stages due to technological constraints imposed by the process. Moreover, each test sample must pass through the pipetting stage in a good time to ensure the validity of tests. For this reason, test samples are initialized in the order of their arrival at the pipetting stage. The laboratory is interested in increasing the number of samples processed during the day. Therefore, effective scheduling of the operations at the pipetting stage is required in order to provide their fast completion and to avoid unnecessary delays and idle times on the machines.

Pipetting involves inoculating small portions of test samples into test plates containing microbiological media, and some auxiliary operations. There are several identically equipped pipetting stations in the laboratory, working in parallel. The diluted test samples arrive to the pipetting stage in plastic bags with barcode labels. The barcode contains essential information about the test suite of a test sample. The request for test plates for a test sample is initiated on one of the pipetting stations by means of scanning the barcode label on the test sample bag. Such a request is automatically transmitted to the plate labelling machine connected to all the pipetting stations. The labelling operation is fully automated. Specifically, it collects from the plate storage desk the required number of test plates of different types in accordance with the test suite of a sample, puts a barcode label containing information about individual test onto each plate, and delivers a stack of labelled plates to the pipetting station which has initiated the request for plates. The pipetting operation may then commence on the corresponding pipetting station.

The plate labelling machine processes incoming requests on the first-come-first-served basis. Therefore, if a pipetter wishes to get on with the testing allocated to her, she will scan several bags at a time. This practice may, however, delay the work of other pipetters, and is therefore disapproved of. The recommended working practice is for each pipetter to scan one bag at a time. The next bag can be scanned by a pipetter only after the pipetting operation for the previous bag is completed.

To summarize, we note that the pipetting process consists of three consecutive operations, namely initialization, plate labelling and pipetting. The first and third operations are performed on the same pipetting station and the second operation is carried out on the plate labelling machine. The sequence in which jobs commence at the pipetting stage is given, and cannot be changed.

2.2 Mathematical formulation

There are m parallel identical machines M_i, for $i = 1, \ldots, m$ which we refer to as *primary* machines. In addition, a single machine S serves the primary machines. Each job j from the set $N = \{1, 2, \ldots, n\}$ consists of three operations: *initialization*, that can be done by any primary machine M_i and requires a small amount of time δ; *setup* operation, performed by machine S during s_j time units; and the *main processing* operation, with processing time p_j and completed by the same machine which has initialized the job. The main processing operation of a job can start only after its initialization and setup operations are completed. Each machine and server S can handle at most one job at a time and each job can be assigned to at most one machine. Each job requires the server between its initialization and processing operations.

Throughout the paper we assume that initialization time is always smaller than any setup time and any processing time, i.e.,

$$\begin{aligned} \delta &\leq \min_{j \in N} \{s_j\}, \\ \delta &\leq \min_{j \in N} \{p_j\}. \end{aligned}$$

For 2-machine case, i.e. when $m = 2$, it is convenient to refer to machine M_1 and M_2 as A and B, respectively. Due to the technological restrictions, initialization and setup operations must be performed in a given, prescribed order, which we will label $(1, 2, \ldots, n)$. Thus, for any pair of jobs j and k, $j < k$, job j is initialized before job k and the setup operation of j is performed before the setup operation of k.

Note that it is possible to initialize several jobs in succession from the same primary machine thus forming a *batch* of jobs. The setup operations of the jobs from a batch are performed consecutively on machine S in the order of their initialization. A batch is *consistent* if no operation of a job from another batch is performed on the primary machine in between the initialization operations and the main processing operations of the jobs from the batch. In other words, consistent batching means that a machine cannot start another job or a batch of jobs before all operations of the currently processed batch are fully completed. Consistent batching reflects the operational requirements in the microbiology laboratory, therefore we shall only consider the class of schedules with consistent batches throughout the rest of the paper. Inconsistent batching is briefly discussed in Appendix A.

An example of a feasible schedule with 9 jobs and 4 batches is given in Fig. 1. To distinguish between the jobs initialized by different machines, we shade the jobs initialized

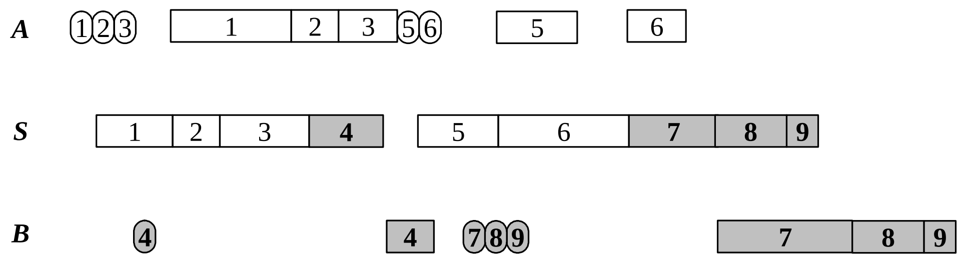

Figure 1: An example of a feasible schedule with 9 jobs

on B. Initialization operations are shown as oval shapes to distinguish them from the main processing operations within rectangular boxes.

The throughput considerations discussed above are encapsulated in the objective of minimizing makespan. The makespan, $C_{\max}$, is determined as the latest completion time of any job:

$$C_{\max} = \max_{1 \le j \le n} \{C_j\},$$

where C_j is the completion time time of job j. Observe that the makespan is not necessarily delivered by job n.

A schedule can be specified either by the start times or the completion times of each operation, and we adopt the following notation. Initialization of job j starts at time $I_j^{M_i}$ if machine M_i is selected for processing; starting and competition times of the setup operation for job j are denoted by T_j^S, C_j^S and those of the main operation by $T_j^{M_i}, C_j^{M_i} = C_j$, if machine M_i initializes job j.

3 The computational nature of the ISP problem

In this section, we explore the computational structure of our ISP problem. We first establish that the problem is NP-complete in the ordinary sense, even for 2 primary machines. It is therefore sensible to tackle the problem with a heuristic approach. Before developing such an approach in the following sections, we examine the structure of the problem in more depth. In subsection 3.2 we establish the relationship of our problem to the reentrant chain scheduling problem. With 1 primary machine in the ISP problem, the setting is the same as that of the 2-machine chain-reentrant shop problem. The machine assignment aspect is trivial, leaving only the sequencing of jobs, which is the central issue for reentrant chains. Thus, the 2 primary machine case is the smallest of interest, and is itself hard without any consideration of sequencing. We then explore differences and similarities with variants of the problem, and in doing so establish some structural properties of an optimal solution. We demonstrate that splitting the jobs into batches assigned to alternative machines at the initialization stage, determines the structure of an optimal schedule at the next two stages, namely setup and main processing. The remainder of the paper then focuses on a heuristic for the 2 primary machine case of the ISP problem, informed by the structure of an optimal solution.

3.1 NP-hardness

Theorem 1 *Problem ISP is NP-hard in the ordinary sense.*

Proof. We construct a reduction from the PARTITION problem which is known to be NP-complete: given z different positive integers $e_1, \dots, e_z$ and $E = 1/2 \sum\limits_{i=1}^{z} e_i$, do there exist two disjoint subsets N_1 and N_2 such that $\sum_{i \in N_1} e_i = \sum_{i \in N_2} e_i = E$? The reduction is based on the following instance of the decision version of problem ISP with $m = 2$, and $n = z$ jobs and processing times

$$\delta = \varepsilon, \quad s_j = \varepsilon, \quad p_j = e_j, \quad j = 1, \dots, z.$$

In such an instance, the initialization and setup times for all operations are the same and equal to a small constant ε. We assume that

$$\varepsilon < \frac{1}{20n}.$$

We show that there exists a schedule σ^* with

$$C_{\max}(\sigma^*) \leq E + 0.1 \tag{1}$$

if and only if the PARTITION problem has a solution.

Suppose PARTITION problem has a solution. Due to the choice of ε, in any feasible schedule, if job j is the first one in a batch initialized on machine A, then

$$I_j^A + \varepsilon = T_j^S \text{ and } T_j^S + \varepsilon \leq T_j^A.$$

Similar conditions can be formulated for each first job initialized on machine B.

Let a subset of jobs $N_1 \subset N$ be initialized on A, and the remaining jobs $N_2 = N \backslash N_1$ initialized on B. Then the completion times of the last jobs on A and B satisfy the relations:

$$\alpha \leq \sum_{i \in N_1} p_i + 2\left|N_1\right| \varepsilon < \sum_{i \in N_1} p_i + 2n\varepsilon$$

and

$$\beta \leq \sum_{i \in N_2} p_i + 2\left|N_2\right| \varepsilon < \sum_{i \in N_2} p_i + 2n\varepsilon,$$

see Fig. 2.

Due to the choice of ε,

$$C_{\max} = \max\{\alpha, \beta\} < \max\left\{\sum_{i \in N_1} p_i, \sum_{i \in N_2} p_i\right\} + 2n\varepsilon = E + 0.1.$$

Suppose now that PARTITION problem does not have a solution. Then for any splitting of the set N into two sets N_1 and N_2,

$$\max\left\{\sum_{i \in N_1} p_i, \sum_{i \in N_2} p_i\right\} \geq E + 1.$$

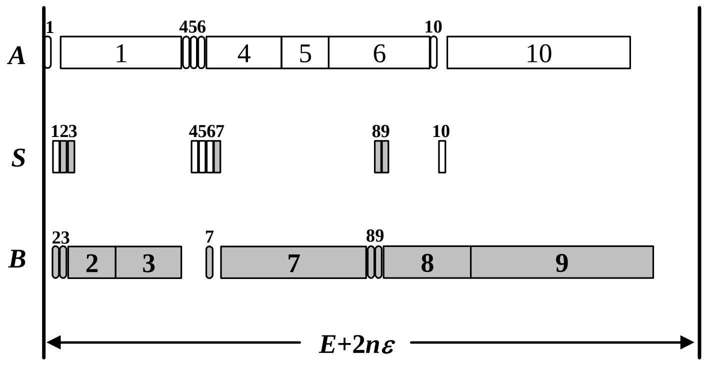

Figure 2: A schedule with $C_{\max} \le E + 2n\varepsilon$

It follows that

$$C_{\max} = \max\{\alpha, \beta\} > \max\left\{\sum_{i\in N_1} p_i, \sum_{i\in N_2} p_i\right\} \ge E + 1,$$

i.e., no schedule σ^* which satisfies (1) exists. ■

3.2 Reentrant aspect of the problem

Wang et al. [8] and Drobouchevitch and Strusevich [9] considered chain-reentrant shop problem with two machines (i.e. one primary machine and one secondary machine) and provided heuristic algorithms with a worst-case performance guarantee of 3/2 and 4/3, respectively. The problem structure for ISP when $m = 1$ is the same as the 2-machine chain-reentrant shop problem. In this restricted case, it is always best to schedule all the first operations before all of the third operations on the single, primary, machine, M, if makespan is to be minimized. In addition, the permutation schedule, in which the first and third operations on M both appear in the same order as the operations on the secondary machine, S in our notation, is always best. Thus, there is no batching or assignment decision to make, and ISP is trivial, while the sequencing problem is not. On the other hand, for our model with two primary machines, two tasks can be considered. Apart from conventional task of determining the sequence of jobs which minimizes makespan, the problem of allocating jobs to primary machines for a given sequence of jobs deserves separate investigation. We will concentrate on its analysis in this paper.

A job can be initiated by the machine at any time when the machine is idle and the initialization time is small, so it is possible to initiate a number of jobs in succession from the same machine, thus creating a *batch* of jobs to be processed on the machine. A detailed review of scheduling problems with batching can be found in Potts and Kovalyov [10]. However, we use the term "batch" in a different context from classical problems with batching.

3.3 Structure of an optimal solution

Property 1 *There is always an optimal schedule for the 2-machine ISP problem with the following properties:*

(i) *the batches are initialized by alternative machines, so that if jobs in the h batch, $\mathcal{B}_h$, are initialized on A, then the $h+1$ batch, $\mathcal{B}_{h+1}$, is initialized on B or vice versa;*

(ii) *for each batch, initialization, setup and main processing operations of jobs are sequenced in the same order.*

Proof. To prove property (i), consider an optimal schedule in which the same machine (say, machine A) initializes two consecutive batches, $\mathcal{B}_h$ and $\mathcal{B}_{h+1}$ say. It is sufficient to prove that moving initialization operations of $\mathcal{B}_{h+1}$ immediately after the initialization of $\mathcal{B}_h$ does not postpone the completion time of the last operation of $\mathcal{B}_{h+1}$ on the corresponding primary machine A, nor of the completion time of the last operation of $\mathcal{B}_{h+1}$ on the server. Observe that the server is idle between batches $\mathcal{B}_h$ and $\mathcal{B}_{h+1}$ for at least the processing time of the last job of batch h and the initialization of batch $h+1$. Merging initialization operations of $\mathcal{B}_h$ and $\mathcal{B}_{h+1}$ eliminates the idle interval on the setup machine S caused by batch $\mathcal{B}_{h+1}$, i.e. the setup operation of the first job in batch $\mathcal{B}_{h+1}$ is performed immediately after the setup operation of the last job in $\mathcal{B}_h$. Moreover, completion of the two sets of initialization operations, and processing of $\mathcal{B}_h$, on machine A is not delayed. Thus, the revised schedule is also optimal. By repeating this process we arrive at an optimal schedule which satisfies property (i).

We now prove property (ii) for a fixed batch $\mathcal{B}_h$ initialized on machine A; the case of the alternative machine B is similar. Recall that the sequence in which jobs are initialized is given and we index the jobs in that order. In addition, setup operations are performed in the same order. Consider the last two stages of processing the subset of jobs $\mathcal{B}_h$, setup and main processing. Machine A processes the jobs from $\mathcal{B}_h$ after all initialization operations of these jobs are completed on A and before the next batch $\mathcal{B}_{h+2}$ is initialized on that machine. The subproblem of scheduling the jobs from $\mathcal{B}_h$ on the main processing machine A after the setup operation is done on machine S, therefore reduces to the two machine flow-shop scheduling problem with machines S and A and the fixed order of jobs on machine S. Since for the two-machine flow-shop problem there always exists an optimal permutation schedule with the same order of jobs on S and A [11], we conclude that the jobs from $\mathcal{B}_h$ should be sequenced on the main processing machine in index order. ■

To illustrate property (i) consider the example of the schedule given in Fig. 3(a). The earlier batches $\mathcal{B}_1 = (1,2,3)$, $\mathcal{B}_2 = (4,5)$, $\mathcal{B}_3 = (6)$, $\mathcal{B}_4 = (7)$, $\mathcal{B}_5 = (8,9)$ are initialized by alternative machines A, B, A, B, A, respectively, while the last batch $\mathcal{B}_6 = (10)$ is initialized by the same machine A, as is the previous batch $\mathcal{B}_5$. Fig. 3(b) demonstrates that moving initialization operation of job 10 from $\mathcal{B}_6$ to immediately after the initialization operations of $\mathcal{B}_5 = (8,9)$ reduces the completion time of the main processing operation of job 10.

It is instructive to observe that the quality assurance constraint of consistent batches on each primary machine may be detrimental to makespan, as illustrated in Appendix A.

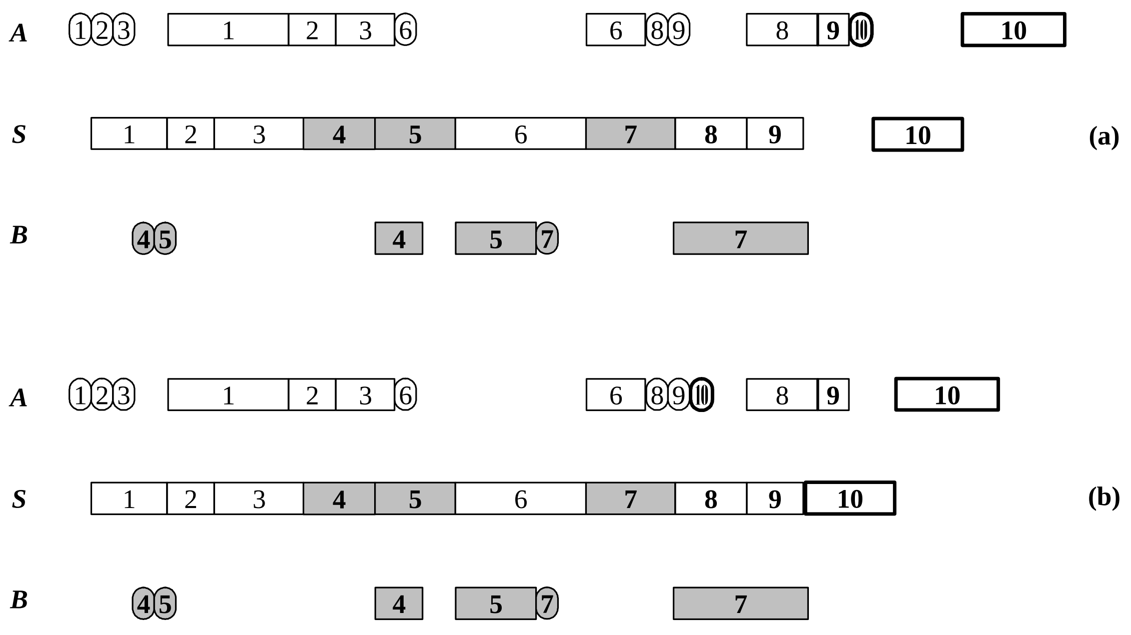

Figure 3: An example of the effect of merging batches: the last three jobs on machine A processed (a) as consecutive batches $\mathcal{B}_5 = (8, 9)$ and $\mathcal{B}_6 = (10)$ and (b) as a merged batch $\mathcal{B}_5 = (8, 9, 10)$

3.4 An implicit representation of an optimal schedule

The structural properties of an optimal solution, highlighted in Property 1, enable us to provide a very succinct representation of an optimal solution. This representation, and its realization as a schedule, are both described below.

An optimal solution is thus defined by an assignment of jobs, or more precisely of batches, to primary machines. The subset, $\mathcal{J}_1$, of jobs which are processed on a different machine from its predecessor, thus identifies a solution. The corresponding schedule is defined by the starting times of initialization and processing of jobs on machine A and B. We now show how the starting times of all operations of a batch can be calculated. Since the two primary machines, A and B, are identical, solutions arise in pairs with the same job completion times. We restrict attention to schedules in which job 1 is assigned to machine A. Now consider a batch consisting of jobs $j, j+1, \ldots, j'-1$, where $j' \in \mathcal{J}_1$ and is processed on machine B. Let machines A, B and S complete processing the preceding jobs $1, 2, \ldots, j-1$ at the time instances α, β and γ, respectively. Start with $\alpha = \beta = \gamma = 0$. Then the starting times of the three operations of the first job j of a batch are given by the formulae:

$$\begin{aligned} I_1^A &= 0, &(2)\\ I_j^A &= \max\left\{I_{j-1}^B + \delta,\ \alpha\right\}, j \in \mathcal{J}_1, &(3)\\ T_j^S &= \max\left\{I_j^A + \delta,\ \gamma\right\}, &(4)\\ T_j^A &= \max\left\{T_j^S + s_j,\ I_j^A + \left(j' - j\right)\delta\right\}, &(5)\end{aligned}$$

where j' is the next job after j in $\mathcal{J}_1$, and hence $(j'-j)\delta$ corresponds to the total initialization time of the jobs of the current batch.

For the remaining jobs ℓ of the batch, $j+1 \le \ell \le j'-1$, their starting times can be determined as follows:

$$\begin{aligned} I_\ell^A &= I_j^A + (\ell - j)\delta, & (6)\\ T_\ell^S &= \max\left\{I_\ell^A + \delta,\ T_{\ell-1}^S + s_{\ell-1}\right\}, & (7)\\ T_\ell^A &= \max\left\{T_\ell^S + s_\ell,\ T_{\ell-1}^A + p_{\ell-1}\right\}. & (8)\end{aligned}$$

Observe that completion times are given by

$$\begin{aligned} C_j^A &= T_j^A + p_j, & (9)\\ C_j^S &= T_j^S + s_j. & (10)\end{aligned}$$

Similar formulae hold if batch $\{j, j+1, \ldots, j'-1\}$ is processed on machine B, except for batch 2, which is the first batch on machine B. The starting time of initialization operation of the first job in this batch, j, which is the second element in $\mathcal{J}_1$, is given by

$$I_j^B = (j-1)\delta, \qquad (11)$$

since the initialization operations are carried out in sequential order, independent of the primary machine.

4 Algorithms

In this section we describe our proposed heuristic algorithm, *CS*, for solving the ISP problem, along with the algorithms which we use for benchmarking purposes. The no-batching algorithm, NB, which is used in practice, is outlined in section 4.3. This provides a good practical comparator, to illustrate the improvement made by introducing our heuristic, *CS*. While a Branch and Bound optimisation method, described in section 4.4, provides a theoretical best case. We start by describing the greedy heuristic, *Basic*, which lies at the heart of our heuristic.

4.1 Basic batching strategy, *Basic(i)*

The algorithm described below constructs a schedule which does not have idle time on the labeling machine S. It aims at balancing the workloads of the main processing machines A and B by coordinating completion times of jobs on these two machines as much as possible. Balancing of machines A and B in our algorithm is achieved by splitting the jobs into batches which are small enough for no idle time to incur on machine S. The schedule is constructed by adding jobs one by one to the partial schedule and batching decisions are based on times α, β and γ when the last job on machines A, B and S in the partial schedule, respectively, is completed.

The algorithm starts with scheduling first job on machine A. Then in each iteration of the algorithm a new job is added to the constructed partial schedule either at the end of the previous batch or as the first job of a new batch. Scheduling at the end of a batch

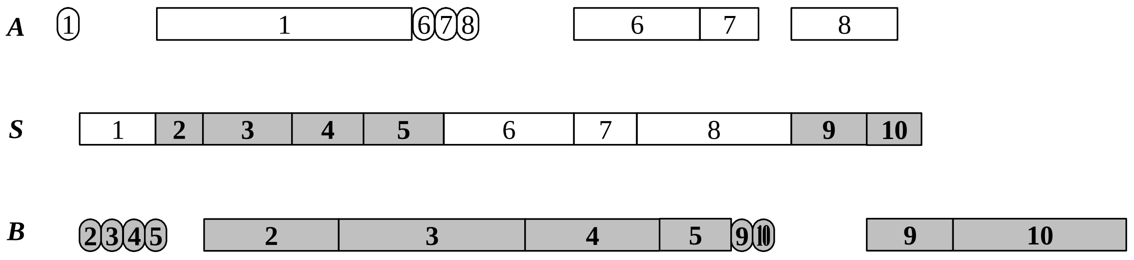

Figure 4: An example of a schedule constructed by Algorithm *Basic(1)*

never leads to an idle time on S while initializing a new batch may lead to an idle time on machine S if the machine that initializes a new batch becomes free later than machine S. In order to make the batches as small as possible, preference is given to initiating a new batch at an earliest opportunity when no idle time on S is created. Observe that the decision to complete a batch and change primary machine depends only upon the relative completion times of the previous batch on the other machine and the last job of the current batch on the server. The timing of processing operations of jobs may therefore be delayed and done for all jobs in the batch after the batch is completed.

We present a pseudocode of algorithm *Basic(1)* and show that its runtime is linear in the number of jobs, n, in Appendix B. An example of a schedule constructed by algorithm *Basic(1)* is illustrated in Fig 4.

Despite its simplicity, algorithm *Basic(1)* can provide considerable improvement over the default approach of single job alternating batches (algorithm *NB*, described in Subsection 4.3). In fact, *Basic(1)* can be shown to provide a relatively good solution under all circumstances. To be more precise, its solutions are guaranteed to be within a factor of 2 of the optimal makespan as we prove in Appendix C.

A drawback of algorithm *Basic(1)* is that it constructs a schedule in which the first batch containing only one job. However, scheduling a single job in the first batch is not necessarily the best option. A larger first batch is better for many instances of the problem. We, therefore, modify algorithm *Basic(1)* to accommodate different sizes of the first batch. We refer to the modified version of the algorithm as *Basic(i)*, where i denotes the number of jobs in the first batch, $i \in \{1, 2, \ldots, n-1\}$. Algorithm *Basic(i)* uses the same batching strategy as algorithm *Basic(1)* for jobs $i+1$ onwards. We denote the resulting schedule and its makespan as $\sigma^{(i)}$ and $C_{\max}\left(\sigma^{(i)}\right)$, respectively. A pseudocode for algorithm *Basic(i)* can be found in Appendix B.

4.2 Proposed heuristic algorithm, *CS*

The heuristic which we propose, *CS*, is based upon the greedy algorithm *Basic* described in the previous subsection. Note that algorithm *Basic(i)* takes the number of jobs in the first batch as an input. However, it is difficult to determine analytically the size of the first batch in an optimal schedule. We therefore incorporate a search over the solution space involving all possible sizes of the first batch in our heuristic algorithm. Algorithm *CS* presented below returns the best schedule σ^{CS} obtained after running algorithm *Basic(i)* and applying a repair operation to the schedule obtained for each value of i, $i = 1, 2, \ldots, n-1$. The repair operation is associated with the end of a schedule and is motivated and described below.

Algorithm CS (Coordinated Scheduling)

```
1. Set $C^{CS} = M$, where $M$ is a large integer.
2. For $i = 1$ to $n - 1$
    2.1. Apply algorithm Basic(i) to construct schedule $\sigma^{(i)}$ with $i$ jobs in the first batch.
    2.2. Apply algorithm Repair to schedule $\sigma^{(i)}$ to obtain schedule $\sigma^{[i]}$.
    2.3. If $C_{\max}\left(\sigma^{[i]}\right) < C^{CS}$, set $C^{CS} = C_{\max}\left(\sigma^{[i]}\right)$, $\sigma^{CS} := \sigma^{[i]}$.
   EndFor
3. Return $\sigma^{CS}$, $C^{CS}$.
```

The example of a schedule produced by algorithm *Basic(1)* (see Fig. 5, schedule (a)) shows that avoiding idle time on machine S may lead to unbalanced workloads on machines A and B. Similar examples can be presented for other values of i in algorithm *Basic(i)*. Such a situation is likely to arise when processing times p_j dominate setup times s_j. Observe that the balance in workloads is actually broken in the tail of the schedule, when the last batch on one of the machines is completed significantly later than the last batch on another machine. For that reason, we introduce another modification of *Basic(i)*, a repair operation, whose goal is to improve such a balance by creating a single idle interval on server S and rearranging the jobs from the last one or two batches between machines A and B.

The tail of a schedule obtained by algorithm *Basic(i)* can be reconstructed as follows. We define the *critical batch* as the batch whose last job completes at $C_{\max}$ in the schedule constructed by algorithm *Basic(i)*. Let job $j_{cr}^{(i)}$ be the first job in a critical batch. Denote by $\sigma_{cr}^{(i)}$ the partial schedule in which jobs $1, 2, \ldots, j_{cr}^{(i)}$ are scheduled by algorithm *Basic(i)*. In order to achieve a better balance of workload between machines A and B and improve upon the performance of algorithm *Basic(i)*, we reschedule jobs $j_{cr}^{(i)}+1, \ldots, n$ by "breaking" a critical batch in a certain point. Specifically, we schedule a job ℓ, $j_{cr}^{(i)} < \ell < n$, from a critical batch as the first job of a new batch processed on the alternative machine, thus creating idle time on server S before the setup operation of job ℓ is started. Then we continue to construct the schedule $\sigma^{[i]}$ by applying the strategy described in algorithm *Basic(1)*. A description and time complexity of algorithm *Repair* which performs the search for schedule $\sigma^{[i]}$ with a minimum makespan, by breaking the critical batch in different points, are given in Appendix B.

To demonstrate the performance of algorithm *Repair*, consider the instance of the problem with $n = 10, \delta = 1, \mathbf{s} = (s_j) = (3,4,2,3,2,1,2,2,2,4)$ and $\mathbf{p} = (p_j) = (5,8,5,7,6,4,6,5,5,7)$. Fig. 5 shows the schedule constructed by algorithm *Basic(1)* with $C_{\max} = 48$ (see schedule (a)) and the schedule with $C_{\max} = 41$ produced by applying the repair operation (schedule (b)).The critical batch of the original schedule $\sigma^{(1)}$ contains six jobs $4, 5, 6, 7, 8$ and 9 processed on machine A. This batch is broken after job 5 in schedule (b). This creates idle time on machine S between the completion of setup operation for job 5 and the start of setup operation for job 6 in schedule (b). On the other hand, the

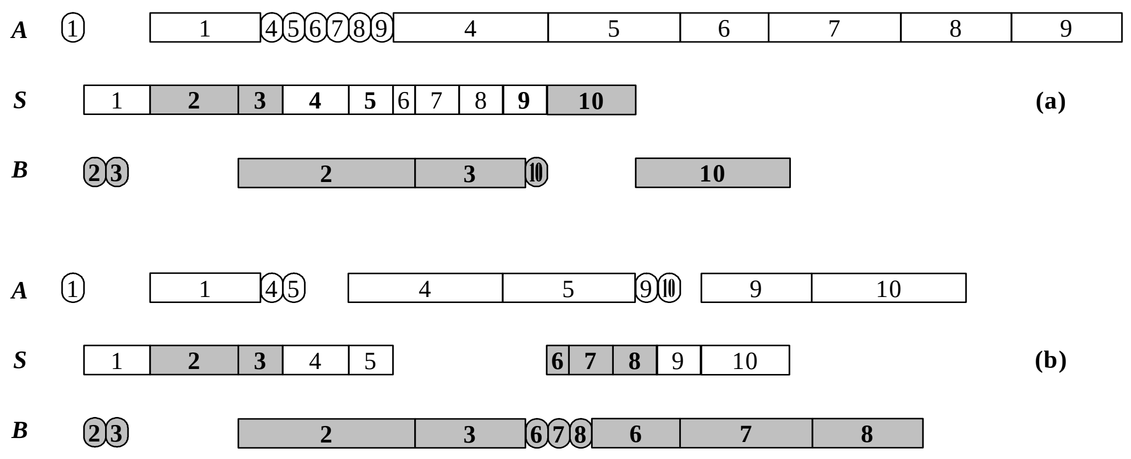

Figure 5: (a) Schedule constructed by algorithm *Basic(1)* and (b) improved schedule obtained by algorithm *Repair*

workloads of machines A and B are better balanced in schedule (b) than in schedule (a), which leads to an improved makespan value.

The time complexity of algorithm *CS* is $O(n^3)$ since algorithm *Repair* is called $n-1$ times at step 2 and itself has complexity $O(n^2)$ as shown in Appendix B.

4.3 Default, no batching, algorithm

In order to show the efficiency of the batching strategy of algorithm *CS,* we need to compare it to a simple "no-batching" procedure which is currently used in practice. The "*no-batching*" *strategy* assumes that a new job can not be initialized on a machine until the previous job assigned to that machine has been completely processed. Each consecutive job is initialized and then processed on the machine which becomes available first. The resulting schedule is denoted by σ^{NB}.

Algorithm NB (No Batching)

```
1. Initialization: $\alpha = \beta = 0$.
2. Schedule three operations of job 1 on machines $A$, $S$ and $A$, respectively. Set $\gamma = \delta + s_1$, $\alpha = C_1^A$.
3. For $j = 2$ to $n$
    If $\alpha > \beta$ then
        Schedule three operations of job $j$ on machines $B$, $S$ and $B$, respectively.
        Set $\gamma = C_j^S = \max\{\beta + \delta, \gamma\} + s_j$, $\beta = C_j^B = C_j^S + p_j$.
    else
        Schedule three operations of job $j$ on machines $A$, $S$ and $A$, respectively.
        Set $\gamma = C_j^S = \max\{\alpha + \delta, \gamma\} + s_j$, $\alpha = C_j^A = C_j^S + p_j$.
```

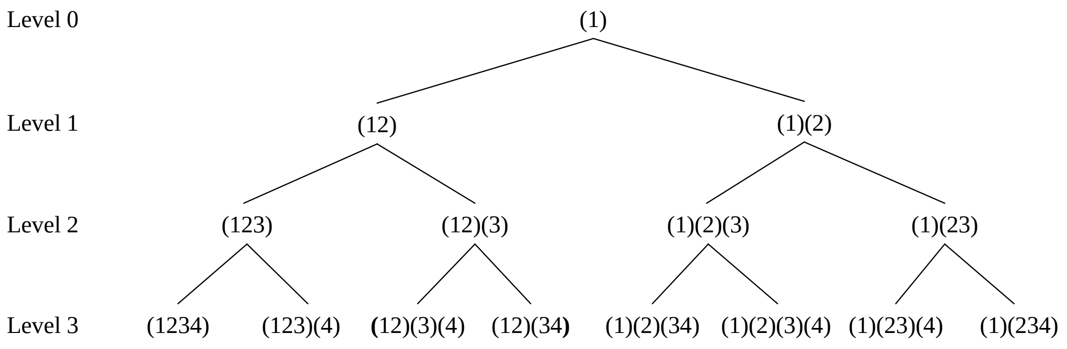

Figure 6: A branching tree for a set of 4 jobs

```
    EndFor
4. Return $\sigma^{NB}$, $C_{\max}(\sigma^{NB}) = \max\{\alpha, \beta\}$.
```

Note that the no batching procedure described above can also be interpreted as a sequential batching strategy with a constant batch size of 1. This strategy shares the feature of algorithm *CS* of taking the first available machine for each batch assignment. Observe that there is an even simpler strategy, in which jobs are assigned to alternative machines. Such a strategy can never do better than *NB*, and when processing times vary is likely to do considerably worse.

4.4 Optimization, Branch and Bound, algorithm

To assess the performance of algorithm *CS*, we develop the branch-and-bound algorithm which implicitly enumerates all possible schedules for a given instance of the problem ISP and finds a solution with a minimum makespan. The branching tree represents a binary tree where each node at level j of the branching tree, for $j = 0, 1, 2, \ldots, n-1$, corresponds to a partial schedule of processing the subset of jobs $\{1, 2, \ldots, j+1\}$. Figure 6 shows the branching tree for problem ISP with 4 jobs. The left descendant of each parent corresponds to assigning the next job to machine A, the right descendant relates to scheduling on machine B. Each set of job numbers in parentheses represents a separate batch. For example, node (12)(3)(4) corresponds to the schedule containing 3 batches: jobs 1 and 2 are assigned to machine A in one batch, jobs 3 and 4 form two separate batches on machines B and A, respectively.

The upper bound in the root node of the tree is set to the value of makespan obtained by running algorithm *CS*. This value is compared to the following two conventional lower bounds in order to check the solution of algorithm *CS* for optimality:

$$LB1 = \delta + \sum_{j=1}^{n} s_j + p_n,$$

$$LB2 = \frac{1}{2}\sum_{j=1}^{n}(p_j + \delta).$$

$LB1$ takes into account the total load of server S and $LB2$ is based on equal workloads of machines A and B.

For all other nodes of the tree, we calculate two lower bounds in the following way. Let α_{jk}, β_{jk} and γ_{jk} be the completion times of the last jobs on machines A, B and S, respectively, in a partial schedule for the first $j+1$ jobs in node k at level j of the tree, $k = 1, 2, \ldots, 2^j$. The values α_{jk}, β_{jk} and γ_{jk} can be calculated using recursive equations (3)-(8) given in subsection 3.4. The first node related lower bound, $LB3_{jk}$, is calculated using the assumption that setup operations of all remaining $n-j-1$ jobs are performed on server S without delays, i.e.

$$LB3_{jk} = \gamma_{jk} + \sum_{i=j+2}^{n} s_i + p_n.$$

The second node specific lower bound, $LB4_{jk}$, provides the earliest possible completion times of the main processing operations of the remaining jobs on machines A and B by distributing the total remaining workload between the two machines:

$$LB4_{jk} = \max\left\{\alpha_{jk}, \beta_{jk}\right\} + \frac{1}{2}\left(\sum_{i=j+2}^{n}(p_i + \delta) - |\alpha_{jk} - \beta_{jk}|\right).$$

The maximum value among $LB3_{jk}$ and $LB4_{jk}$ is selected as a lower bound associated with the node. A node is discarded without further expansion if its lower bound is no less than the current upper bound. The upper bound is updated if a leaf node of the tree is reached and its makespan value is lower than the current upper bound value.

We use a *min heap* data structure to maintain the list of the nodes which are not discarded in the process of their generation. Min heap ensures that the node with a smallest value of the lower bound, i.e. the node potentially leading to an optimal solution, will be taken out from the list for further expansion at each step of the branch and bound algorithm. The algorithm stops when the list of nodes for expansion becomes empty. The final upper bound value corresponds to the optimal makespan.

5 Computational experiments

In this section we evaluate the performance of algorithm *CS* on a broad range of problem instances. We will show that the batching strategy of our algorithm is significantly more efficient that a simple "no-batching" procedure described in Subsection 4.3 which is currently used in practice. In addition, we evaluate the absolute quality of our algorithm by comparison with an optimal solution provided by a branch and bound algorithm discussed in Subsection 4.4. We test algorithm *CS* on different patterns of randomly generated data described in Subsection 5.1 and analyze the results. We first present the initial experiments which we carried out to establish the dimensionality of the problem, in Subsection 5.2. Later, in Section 5.4, we will demonstrate the advantage of our approach for the microbiology laboratory which inspired our research.

5.1 Experimental design

We created a benchmark dataset of problem instances upon which to test out our algorithm. To do so we considered a wide range of values for parameters which are likely to be significant for scheduling strategies. The relative sizes of the three processing times of a sample, for initialization, setup and processing, are obviously relevant, as is the distribution of times between samples. The number of jobs is yet another dimension.

Before proceeding to a full set of computational experiments, we investigated two aspects of the problem instances in order to identify their significance. The first one relates to the times of setup operations on the server and of main processing operations on a primary machine. The second aspect concerns short initialization time on the primary machine. These initial experiments described below indicate which parameters are significant for the full set of computational experiments and hence how best to group problem instances.

For our main experiments, we consider instances of the problem ISP with $n = 10, 25$ and 50 jobs. For each value of n, 9 separate patterns of data are tested. For each pattern, we generate 50 problem instances using different random seeds. We choose the ranges from which values of s_j and p_j are generated in such a way as to ensure that the average ratio between workloads on the server and processing machines over 50 instances approaches the corresponding value of R. Initialization time δ is standardized at value 1. The initial experiments indicate that no further dimensionality is required. Table 1 presents a summary of input test data.

Pattern	R	s_j range	p_j range
1	0.25	[5,35]	[120,200]
2	0.5	[5,35]	[60,100]
3	0.75	[5,35]	[36,70]
4	1	[5,35]	[20,60]
5	1.25	[5,35]	[17,47]
6	1.5	[5,35]	[11,41]
7	1.75	[5,35]	[8,38]
8	2	[5,35]	[5,35]
9	2.5	[5,35]	[5,27]

Table 1: Summary of generated data

Note that different patterns in Table 1 define different correlations between values of s_j and p_j for individual jobs. For the first 3 patterns with lower values of R, we can observe a clear dominance of processing operations over setup operations for all jobs which makes the processing stage a bottleneck in the system. As R increases, the number of jobs for which $s_j > p_j$ grows, and the server becomes a bottleneck. Patterns with $R \geq 1$ represent mixes of jobs with different proportions of dominating setup and processing operations.

All three algorithms tested, namely *NB*, *CS* and branch and bound, are programmed on C++ language and tested on 3GHz PC with 512 MB RAM running under Microsoft Windows XP.

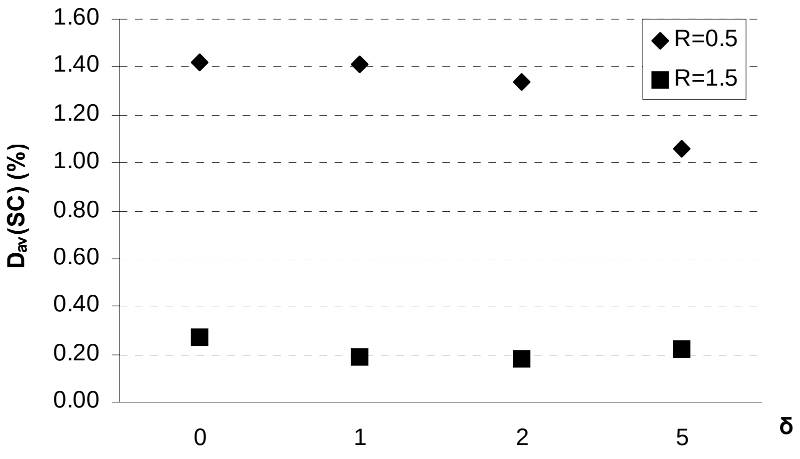

Figure 7: Effect of changing δ on the performance of algorithm *CS*

5.2 Initial experiments

Our initial computational experiments to identify the significance of parameters are described below. We focus upon two aspects of problem instances: the relative times of setup and processing of samples, and the significance of initialization time. The experimental results and conclusions are outlined for each below. All initial experiments were conducted upon the full range of setup and processing times and number of jobs employed in the final benchmark dataset.

We generated datasets in which setup times s_j and processing times p_j are taken from different ranges. Two sets of scenarios were created in which either the setup operations or the main processing operations of jobs were dominant. We also considered a range of "mixed" datasets with different proportions of jobs with dominating setup component and jobs for which main processing operations were longer than their setup operations. The analysis of the problem structure indicated that the relative workload between the primary parallel machines and the setup machine was probably a key factor in determining the potential gain from batching jobs. Our initial computational experiments confirmed this. We therefore selected patterns of data characterized by the value of the ratio R between the total load W_S of server S and average load W_P of processing machines A and B, i.e.

$$R = \frac{W_S}{W_P},$$

where

$$W_S = \sum_{j=1}^{n} s_j \text{ and } W_P = \frac{1}{2}\sum_{j=1}^{n}(p_j + \delta).$$

In our initial experiments for 10 jobs and different values of R, we tested the procedure with no batching (algorithm *NB*) and algorithm *CS* for $\delta = 0, 1, 2$ and 5, keeping setup and processing times of jobs unchanged. The ranges for s_j and p_j were selected in such a way as to ensure that $\min_j \{s_j\} \geq 5$ and $\min_j \{p_j\} \geq 5$. We observed that changing δ does not have a significant impact on the performance of either algorithms *NB* or *CS*.

Figure 7 shows the average performance of algorithm *CS* for 10 jobs, different values of δ and $R = 0.5$ and 1.5. We can notice that algorithm *CS* produces high quality schedules for all tested values of δ. In fact, increasing δ by a certain amount Δ leads to the increase of the makespan values produced by algorithm *NB* by $n\Delta/2$ for most instances of the problem. For both algorithm *CS* and branch and bound the magnitude of makespan increase usually lies in the range $[\Delta, 2\Delta]$. Moreover, increasing δ does not change the structure of schedules (i.e. the allocation of jobs to the machines and their partition into batches) constructed by algorithm *CS* and branch and bound. The reason for this is that δ is small relative to the lengths of setup and processing operations of jobs. Therefore, in what follows we will discuss only the results for $\delta = 1$.

5.3 Results for benchmark datasets

For each pattern of data and each number of jobs, we compare the schedules constructed by the algorithm *NB* to those produced by algorithm *CS*. In addition, we calculate the optimum makespan obtained by branch and bound algorithm described in subsection 4.4. The comparative performance of the algorithms is summarized in Table 2. We use the following notation in the table:

$D_{av}(H)$ – average percentage deviation (over 50 runs) of makespan values produced by algorithm H from the optimal makespans, for $H = NB$ or CS;

$N_{opt}(CS)$ – number of runs (out of 50) when the optimal solutions (or a surrogate in exceptional cases) were found by algorithm *CS*.

The CPU times required for a single run of algorithm *CS* are well under 1 second, while branch and bound algorithm may take long hours for certain runs especially when $n = 50$ and the ratio R is high. Note that the running time of the branch and bound algorithm was therefore restricted to 6 hours. For $n = 50$ and $R > 1$, branch and bound was often unable to finish in such a time limit and the best solutions found in 6 hours were recorded instead of the optimal ones. Therefore, two values, separated by slash, are given in column $N_{opt}(CS)$ in cells corresponding to $n = 50$ and $R > 1$. The first value represents the number of optimal solutions found by algorithm *CS* (either by reaching the lower bound or proven by branch and bound algorithm) while the second one shows the number of runs when branch and bound has not improved solutions found by algorithm *CS* within 6 hours time limit.

Our main observations are the following.

1. *The advantages of batching.* The figures in Table 2 suggest that processing jobs in batches is significantly more efficient than simply assigning a single job to the machine after the completion of the previous job on that machine. Algorithm *CS* produces schedules with makespans which are within the optimal value on average by around 1.0-1.5% for lower values of R and approaching optimal values as the ratio R grows, as indicated by values of $D_{av}(CS)$. On the other hand, "no batching" strategy leads to multiple idle times on all machines, thus completing the processing of given sets of jobs much later than the strategy with coordinated batching used in algorithm *CS*. Schedules obtained by applying algorithm *NB* are 8.5-45.2% worse on average than the optimal schedules with batching (see figures for $D_{av}(NB)$).

Pattern	R	n	$D_{av}(NB)(\%)$	$D_{av}(CS)(\%)$	$N_{opt}(CS)$
1	0.25	10	12.03	0.69	27
		25	13.47	1.06	2
		50	14.00	0.49	2
2	0.5	10	20.85	1.41	25
		25	24.93	1.41	6
		50	25.83	0.86	0
3	0.75	10	26.61	1.27	22
		25	33.87	1.67	8
		50	38.72	1.34	2
4	1	10	29.00	0.80	35
		25	39.03	1.35	15
		50	45.22	1.33	1
5	1.25	10	25.31	0.53	38
		25	35.65	0.70	25
		50	39.26**	0.33**	17/12*
6	1.5	10	19.63	0.19	47
		25	25.80	0.22	40
		50	28.74**	0.00**	28/20*
7	1.75	10	16.89	0.14	46
		25	21.50	0.08	45
		50	23.24**	0.02**	32/16*
8	2	10	14.65	0.06	46
		25	15.51	0.01	48
		50	17.78**	0.02**	40/9*
9	2.5	10	8.56	0.01	49
		25	10.21	0.05	46
		50	11.68**	0.00**	44/6*

* Number of runs when branch and bound did not improve heuristic solution within 6 hours

** The figure is based on complete and terminated runs of branch and bound

Table 2: Summary of the performance of heuristic CS for the benchmark datasets

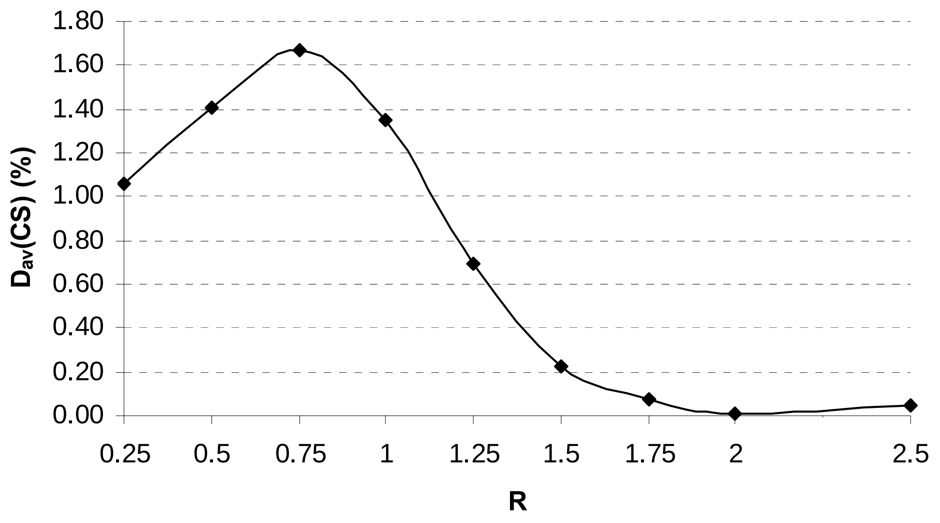

Figure 8: Performance of algorithm CS in relation to relative server workload, for 25 jobs

2. *Effects of the relative workload between server and primary machines, R.* Figure 8 summarizes our analysis and shows the dynamics in results produced by the algorithm as the ratio R raises, for problem instances with 25 jobs. Similar patterns can be observed for 10 and 50 jobs. Note that the "hardest" instances for the algorithm are those corresponding to $R = 0.75$. It appears that if the bottleneck is at the processing stage, algorithm CS performs better for problems with the greater difference between the processing and setup times of jobs (i.e. problems with lower R).

   For $R < 1$, machines A and B are the bottleneck machines. Therefore balancing workloads and avoiding idle intervals as much as possible are the key factors for constructing a good schedule and uninterrupted load of server S is less important. Figures for N_{opt} in Table 2 show that algorithm CS finds optimal solutions only occasionally for instances with a greater number of jobs. This is also much the case for instances with $R = 1$ where there is no clear bottleneck stage and values of $LB1$ and $LB2$ are very close to each other. Our investigation showed that optimal schedules found by branch and bound algorithm often contain a few idle intervals on machine S. On the other hand, algorithm CS only produces schedules with either no idle time or with a single idle interval on machine S at the end of the schedule. Therefore, a schedule constructed by algorithm CS can be potentially improved by splitting large batches and redistributing the jobs between machines A and B which synchronizes completion times of the last jobs on both machines.

   As the ratio R grows beyond 1, the bottleneck moves to machine S and avoiding delays on the server becomes critical. Algorithm CS is designed to fulfil this task in the first instance, therefore the algorithm frequently finds optimal schedules with makespans equal to the corresponding values of lower bound $LB1$. Even when $LB1$ is not reached by our heuristic, branch and bound is often unable to produce a schedule with a better makespan, which provides extra evidence of the strong performance of algorithm *CS*.

3. *Effects of increasing the number of jobs.* Increasing the number of jobs makes the problem ISP more difficult to tackle. Comparing the number of optimal solutions $N_{opt}(CS)$ found by algorithm CS for $n = 10$, 25 and 50 for each value of R, we can observe that $N_{opt}(CS)$ decreases as the value of n is growing.

5.4 Experiments with laboratory data

In this section, we illustrate the advantage of our approach in a real-world setting. We report the results of computational experiments on data provided by the microbiology laboratory. The current practice which predominates at the pipetting stations in the laboratory is that of initializing one sample at a time. This practice is replicated by the "no batching" algorithm, *NB*. We therefore compare the behaviour of our algorithm CS on the given data, to that of algorithm *NB*. The laboratory is equipped with three identical pipetting stations, although generally only two of them are in operation. We therefore run our experiments both for two and three parallel machines.

Historical data reflecting daily demand for testing were provided by the laboratory for 5 different days. A short summary showing the nature of the data is given in Table 3. Since the real sequences in which samples enter the testing process were not recorded by the laboratory, we generated 10 random sequences for each dataset.

Day	Number of test suites	Total number of test samples, n	Workload ratio, R
1	21	573	1.45
2	22	441	1.65
3	31	570	1.58
4	19	502	1.45
5	29	556	1.56

Table 3: Summary of the laboratory data

Each test sample (considered as a job in our problem formulation) arriving to the pipetting stage has to undergo a certain number of microbiological tests which form its test suite. The jobs associated with a given test suite require the same labelling and pipetting times (i.e. setup and processing times, respectively) while these times usually differ between test suites. There are a number of "standard" test suites which are assigned to the majority of test samples entering the process each day. On the other hand, test samples with certain test suites may be absent in some days. Table 3 shows the variety in the number of test suites and in the total number of test samples, n, for 5 days.

The labelling times take values between 23 and 102 seconds, while pipetting times are in the range from 10 to 140 seconds, depending on the test suite. The initialization operation takes 4 seconds. We have calculated the ratios between workloads of the labelling machine and the pipetting stations (assuming that two pipetting stations are used) for each dataset to have a better idea about the nature of the data. In fact, there is a good "mix" of jobs

Day	Total labelling time	Makespan (h:mm)				Savings from batching jobs (%)		
		with no batching		with batching (CS)				
		average	max	average	max	min	average	max
1	5:56	7:47	7:50	5:57	5:58	23.2	23.5	24.1
2	3:47	4:41	4:42	3:47	3:48	18.2	18.9	19.5
3	4:42	5:56	5:57	4:43	4:44	20.3	20.6	20.8
4	5:48	7:46	7:49	5:49	5:50	24.7	25.1	25.6
5	4:40	5:56	5:57	4:41	4:42	20.5	20.9	21.2
Overall						18.2	21.8	25.6

Table 4: Performance of algorithms for two pipetting stations

with labelling times exceeding pipetting times and vice versa in each dataset. Figures from the last column of Table 3 suggest that the labelling machine is a bottleneck. Therefore, we can expect very strong outcomes from algorithm *CS* for such a scenario (see results and discussion in the previous section).

The results of running algorithms *NB* and *CS* for laboratory data and two pipetting stations are summarized in Table 4. All minimum, maximum and average figures in the tables are taken over 10 runs of each algorithm. Due to the large number of jobs in each dataset, running the branch and bound algorithm is impractical. We therefore use lower bounds to assess the quality of schedules. Since labelling machine is a bottleneck, lower bound $LB1$ is most appropriate. The main component of $LB1$ is the total labelling time, which is given in the second column of Table 4.

The advantage of coordinated scheduling strategy with batching lying behind algorithm *CS* over "no batching" strategy is evident from Table 4. The savings for two pipetting stations are in between 1 and 2 hours, averaging 22%. The minimum time savings from using batching recorded in our experiments is about 18% while the maximum savings exceed 25%. Comparing the results of algorithm *CS* with the total labelling time, it can be seen that the difference is just 1 or 2 minutes over a day. This difference is largely accounted for by the pipetting time of the last job. Therefore, the performance of algorithm *CS* cannot, in this context, be improved upon.

The effect of the sequence in which test samples are presented, is indicated by the range of outcomes for the 10 randomly generated sequences within each day's dataset. The data from the laboratory indicate that such a difference is usually under 2 minutes, which is not significant relative to the magnitude of the makespans. Thus, the results suggest that sequencing test samples is not an important issue for the micro-biology laboratory.

Since the labelling machine is a bottleneck, the total labelling time is a crucial factor in determining a good makespan. Our batching strategy gets as near to this makespan as is possible by prioritizing occupancy of the labelling while the conventional no-batching strategy does not. In contrast, the strategy with no batching frequently creates idle times on the labelling machine which leads to significantly longer times required to complete all jobs when 2 pipetting stations are used (see Table 4). Thus this strategy artificially creates a bottleneck at the pipetting stations while the real bottleneck facility remains the labelling machine. The only way of reducing makespan with a no-batching strategy is to deploy an additional pipetter. We therefore also tested *NB* for 3 pipetting stations, against the generalized versions of the *CS* algorithm for completeness. However, algorithm *CS* with 2 pipetters still outperforms algorithm *NB* even with 3 pipetters for all instances of the problem. Indeed, the average time saving is about 3.5% in makespan in addition to the saving of 1 member of staff at the pipetting stations. Note that a single run of algorithm CS on real data takes less than a second for two pipetting stations and does not exceed 1.5 seconds when 3 pipetting stations are used.

The above results illustrate the benefits of the batching strategy of algorithm *CS.* It creates the potential for raising the productivity of the laboratory by over 20%, by accommodating an increased number of test samples processed in the laboratory during a day. Moreover, in current circumstances, the batching strategy should lead to savings in running costs of the laboratory, by eliminating the requirement for a third pipetter at any time.

6 Conclusions

The scheduling problem ISP studied in this paper has been identified at a particular stage of a complex technological process of food testing in the microbiological laboratory. The problem is of a very specific nature. It can be considered as a chain-reentrant two-stage hybrid flowshop with parallel primary machines at the first stage and a single machine at the second stage with the additional requirement that the first and the last operations of each job should be performed by the same primary machine. To the best of our knowledge, hybrid reentrant shop problems have not been studied in the literature before. The closest problems in the literature, which share some common features are the problems of scheduling in a reentrant shop [8] and of scheduling on parallel machines with a common server [1]. We showed that the ISP problem is NP-hard even if the sequence of jobs is fixed. We developed a heuristic algorithm based on a simple strategy of initializing jobs in batches on the primary machines. Our heuristic avoids unnecessary idle times on the machine at the second stage (server) and provides balanced workloads on primary machines. These two features make our algorithm able to find optimal or near optimal schedules in different production scenarios, both when the primary machines and when the server create the bottleneck in the system. We also show that our batching strategy is much more efficient than the strategy with no batching currently used in the laboratory. Adopting batching of jobs in practice may increase productivity of the production line and provide savings in the running costs of the company. Estimates are put at a saving of 1 member of staff or at 22% potential increase in productivity.

Acknowledgements

This research was supported by EPSRC grant number EP/D079713/1. We would like to thank Graeme Jardine, Peter Sadd and Andy Walker of Premier Analytical Services for their cooperation and for providing data for computational experiments.

References

[1] Hall NG, Potts CN, Sriskandarajah C. Parallel machine scheduling with a common server. Discrete Applied Mathematics 2000;102:223-43.

[2] Brucker P, Dhaenens-Flipo C, Knust S, Kravchenko SA, Werner F. Complexity results for parallel machine problems with a single server. Journal of Scheduling 2002;5:429-57.

[3] Kravchenko SA, Werner F. Parallel machine scheduling problems with a single server. Mathematical and Computational Modelling 1997;26:1-11.

[4] Abdekhodaee AH, Wirth A. Scheduling parallel machines with a single server: some solvable cases and heuristics. Computers and Operations Research 2002;29:295-315.

[5] Abdekhodaee AH, Wirth A, Gan H-S. Scheduling two parallel machines with a single server: the general case. Computers and Operations Research 2006;33:994-1009.

[6] Glass CA, Shafransky YM, Strusevich VA. Scheduling for parallel dedicated machines with a single server. Naval Research Logistics 2000;47:304-28.

[7] Morton TE, Pentico DW. Heuristic scheduling systems: with applications to production systems and project management. John Wiley & Sons, New York, 1993.

[8] Wang MY, Sethi SP, van de Velde SL. Minimizing makespan in a class of re-entrant shops. Operations Research 1997;45:702-712.

[9] Drobouchevitch IG, Strusevich VA. A heuristic algorithm for two-machine re-entrant shop scheduling. Annals of Operations Research 1999;86:417-439.

[10] Potts CN, Kovalyov MY. Scheduling with batching: a review. European Journal of Operational Research 2000;120:228-49.

[11] Johnson SM. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1954;1:61-68.

Appendices

A Consistent and inconsistent batching

Consistent batching on a primary machine means that a new batch can not be initialised on a primary machine before all jobs from the previous batch are processed on that machine. A requirement for batch consistency is imposed by the company for quality assurance purposes. In this paper, we consider only the class of schedules with consistent batches.

However, inconsitent batching allows initialized jobs to form a new batch at any time when the primary machine becomes available, e.g. between the processing operations of two consecutive jobs of the previous batch on that machine or before the processing operation of the first job of the previous batch.

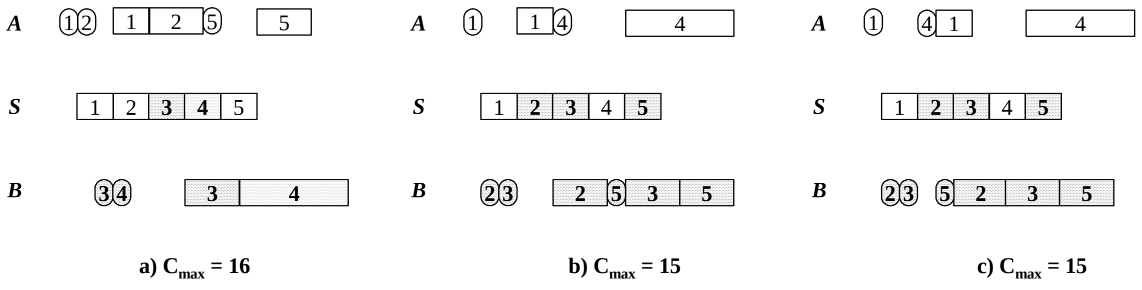

Figure 9: Optimal schedules: a) for the class with consistent batches; b),c) when inconsistent batching is allowed

Consider the instance of the problem with $n = 5$, $\delta = 1$, $\mathbf{s} = (s_j) = (2, 2, 2, 2, 2)$ and $\mathbf{p} = (p_j) = (2, 3, 3, 6, 3)$. Figure 9 shows the optimal schedule for the class of schedules with only consistent batching (Figure 9a) and two examples of optimal schedules for a wider class of schedules where inconsistent batching is allowed (Figures 9b and 9c).In schedule b) job 5 is initialised as the first job of a new batch in between processing operations of jobs 2 and 3 from the previous batch. In schedule c) all initialisation operations for two consecutive batches on each machine are performed before the start of the processing operation of the first job from the first batch on that machine. Observe that both schedules with inconsistent batching have better makespans than the schedule with consistent batching. However, this is not necessarily the case for other instances of the problem. For example, adding job 6 with $s_6 = p_6 = 2$ to the last batch on machine A in schedule a) does not increase the makespan for the new schedule with 6 jobs, and this schedule can not be improved by using inconsistent batching. It would be interesting to determine the conditions which make inconsistent batching advantageous. We leave this question for future research.

B Pseudocodes of algorithm *Basic(1)* and its modifications

In this appendix, we itemize the steps required to implement the basic batching strategy *Basic(1)* and its modifications, *Basic(i)* and *Repair*, described in Section 4. The analysis of time complexity of the algorithms is also presented.

B.1 Algorithm *Basic(1)*

Algorithm *Basic(1)*

```
1. Initialization.
    1.1. Set $\alpha := 0$, $\beta := 0$, $\gamma := 0$. {set machine availability times to zero}
    1.2. Set $j := 1$. {select the first job for scheduling}
    1.3. Assign job $j$ to machine $A$ by scheduling the first two operations of job $j$ (keeping
         the last operation unscheduled until a batch is completed):
             $I_j^A := \alpha, \quad \alpha := \alpha + \delta,$
             $T_j^S := \max\{\alpha, \gamma\}, \quad C_j^S := T_j^S + s_j, \quad \gamma := C_j^S.$     (12)
    1.4. Set $jFirst := 1$. {store the index of the first job in a batch}
2. While $j < n$ {consider each unscheduled job}
    $j := j + 1$.
    2.1. If job $j - 1$ was assigned to $A$ and $\gamma < \beta + \delta$, then
           Add $j$ to the last batch on machine $A$ by scheduling the first two operations of
           job $j$ in accordance with (12) (keeping the last operation unscheduled until a
           batch is completed).
    2.2. If job $j - 1$ was assigned to $A$ and $\gamma \geq \beta + \delta$, then
         2.2.1. {complete the current batch on A by scheduling its last operations}
           For $\ell = jFirst$ to $j - 1$
               $T_\ell^A := \max\{\alpha, C_\ell^S\}, \quad \alpha := T_\ell^A + p_\ell.$
           EndFor
         2.2.2. Assign $j$ as the first job of a new batch on machine $B$ by scheduling the
           first two operations of that job (keeping the last operation unscheduled until
           a batch is completed):
               $I_j^B := \beta, \quad \beta := \beta + \delta,$
               $T_j^S := \max\{\beta, \gamma\}, \quad C_j^S := T_j^S + s_j, \quad \gamma := C_j^S.$
         2.2.3. Set $jFirst := j$.
    2.3. If job $j - 1$ is completed on $B$ and $\gamma < \alpha + \delta$, then
           Add $j$ to the last batch on machine $B$ as for 2.1.
    2.4. If job $j - 1$ is completed on $B$ and $\gamma \geq \alpha + \delta$, then
           Complete previous batch on machine $B$ and start a new batch on machine $A$
           with job $j$ (without incurring an idle time on machine $S$) as for 2.2.
    EndWhile
3. Complete the last batch by scheduling its main processing operations on machine $A$
   or $B$ as for 2.2.1.
4. Return resulting schedule $\sigma^{(1)}$, $C_{\max}\left(\sigma^{(1)}\right) = \max\{\alpha, \beta\}$.
```

The initialization step of algorithm *Basic(1)* consists of a few simple operations and thus requires $O(1)$ time. Within step 2 simple comparisons determine which of the four substeps is invoked at each of $n-1$ iterations. Between steps 2 and 3 each of the n jobs is scheduled. This involves, for each job j, setting I_j^A, C_j^S and T_j^A or I_j^B, C_j^S and T_j^B and resetting at most two of α, β, γ. The setting of each variable involves a simple addition or comparison. Thus overall time complexity of scheduling n jobs is $O(n)$.

B.2 Algorithm *Basic(i)*

```
Algorithm Basic(i)
  1. Initialization.
     1.1. Set $\alpha := 0$, $\beta := 0$, $\gamma := 0$. {set machine availability times to zero}
     1.2. {assign the first $i$ jobs to machine $A$}.
      For $j = 1$ to $i$
           Assign job $j$ to machine $A$ as in step 1.3 of Basic(1).
      EndFor
     1.3. Set $jFirst := 1$. {store the index of the first job in a batch}
 2-3. The same as steps 2 and 3 of algorithm Basic(1).
  4. Return schedule $\sigma^{(i)}$, $C_{\max}\left(\sigma^{(i)}\right) = \max\{\alpha,\ \beta\}$.
```

Similar to *Basic(1)*, algorithm *Basic(i)* has a linear time complexity and worst case performance ratio of 2.

B.3 Algorithm *Repair*

```
Algorithm Repair
   Input: schedule $\sigma^{(i)}$ constructed by algorithm Basic(i)
  1. If $C_{\max}\left(\sigma^{(i)}\right) = \delta + \sum_{j=1}^{n} s_j + p_n$, then STOP (as schedule $\sigma^{(i)}$ is optimal).
  2. Set $C^{[i]} := C_{\max}\left(\sigma^{(i)}\right)$, $\sigma^{[i]} := \sigma^{(i)}$.
     Find the critical batch in schedule $\sigma^{(i)}$.
     Let $j_{cr}$ be the first job in that batch and $q$ be the size of the critical batch.
  3. For $k = 0$ to $q - 2$
     3.1. Start to construct schedule $\sigma$ with the partial schedule of $\sigma^{(i)}$ consisting of jobs
          $1, 2, \ldots, j_{cr}$.
       Schedule the next $k$ jobs $j_{cr}+1, \ldots, j_{cr}+k$ in the same batch as job $j_{cr}$.
     3.2. Schedule job $j_{cr}+k+1$ on the alternative machine, starting a new batch on that
          machine.
     3.3. Construct the remainder of schedule $\sigma$ by assigning remaining jobs
          $j_{cr} + k + 2, \ldots, n$ to the machines $A$ and $B$ using the strategy described
          in steps 2-3 of algorithm Basic(1).
     3.4. If $C_{\max}(\sigma) < C^{[i]}$ then
```

```
        Set $C^{[i]} := C_{\max}(\sigma)$, $\sigma^{[i]} := \sigma$.
    EndFor
4. Return $\sigma^{[i]}$, $C^{[i]}$.
```

Steps 1-2 of algorithm *Repair* require $O(n)$ time. Since the last batch contains $q \leq n-1$ jobs, Step 3 is performed no more than $n-2$ times. In each iteration the $O(n)$-time algorithm *Basic* is called. Thus the overall time complexity of algorithm *Repair* is $O(n^2)$.

C Approximation bound

Theorem 2 *For problem ISP, algorithm Basic(1) has a tight worst case bound of 2.*

Proof. We show that for schedule $\sigma^{(1)}$ constructed by algorithm *Basic(1)* the following inequality holds:

$$\frac{C_{\max}\left(\sigma^{(1)}\right)}{C_{\max}\left(\sigma^{*}\right)} \leq 2,$$

where $C_{\max}(\sigma^*)$ is the makespan of an optimal schedule.

In any schedule there always exists at least one job, called *critical*, which starts on the main processing machine exactly at the time its labeling operation is completed and is scheduled on a machine which achieves the makespan and has no idle time after this critical job. Let k be the latest job in sequence among all critical jobs in the schedule $\sigma^{(1)}$ and let N' be the set of jobs which includes job k and all other jobs scheduled after job k on the same machine. Observe that there can be several batches after the critical job and completion time of some job $n' \in N'$, $n' \leq n$ determines the makespan. Note that algorithm *Basic(1)* produces a schedule in which machine S processes the jobs without idle time. It follows that

$$C_{\max}\left(\sigma^{(1)}\right) \leq \delta + \sum_{j=1}^{k} s_j + \sum_{j\in N'} p_j + \left|N'\right|\delta, \tag{13}$$

where the first term δ corresponds to an idle time on machine S caused by the initialization time of the first job and the last term $|N'|\,\delta$ is the combined length of the initialization operations of the jobs from N'.

We use the following two lower bounds for the makespan of an optimal schedule:

$$\begin{aligned} C_{\max}(\sigma^*) &\geq LB0 = \frac{1}{2}\left(\sum_{j=1}^{n} p_j + n\delta\right), \\ C_{\max}(\sigma^*) &\geq LBk = \delta + \sum_{j=1}^{k} s_j + \frac{1}{2}\left(\sum_{j=k}^{n} p_j + (n-k+1)\delta\right), \end{aligned}$$

where the first inequality is based on the average load of machines A and B, while the second one uses the fact that in any schedule jobs $k, \ldots, n$ cannot start on machines A or B earlier than the time $\delta + \sum_{j=1}^{k} s_j$ and will occupy machine A or B for at least $\frac{1}{2}\left(\sum_{j=k}^{n} p_j + (n-k+1)\delta\right)$. Since $N' \subset \{k, \ldots, n\}$, $LB0 + LBk$ is no greater than the right hand side of (13) and hence

$$C_{\max}\left(\sigma^{(1)}\right) \leq 2C_{\max}\left(\sigma^{*}\right).$$

Therefore *Basic(1)* is a 2-approximation algorithm.

To show that this bound is tight consider the following instance with $n = 3$ jobs: $\delta = \varepsilon$, $s_1 = s_2 = s_3 = \varepsilon$, $p_1 = 1$, $p_2 = T + 1$, $p_3 = T$. Algorithm *Basic(1)* produces the schedule shown in Fig. 10 of length

$$C_{\max}\left(\sigma^{(1)}\right) = 3\varepsilon + 2T + 1,$$

where 3ε corresponds to initialization operation of job 1 and two setup operations of jobs 1 and 2. However, the schedule σ^* illustrated in Fig. 11 has makespan

$$C_{\max}\left(\sigma^{*}\right) = 4\varepsilon + T + 1.$$

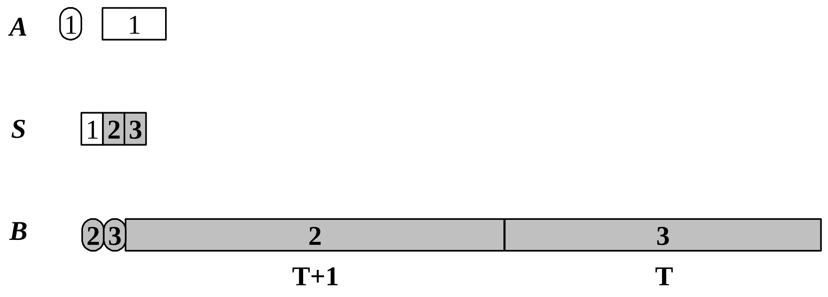

Figure 10: A worst-case instance for algorithm *Basic(1)*

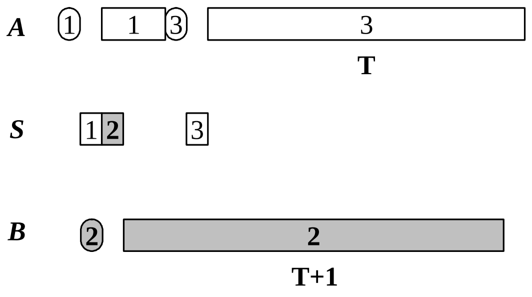

Figure 11: An optimal schedule for the worst-case instance

Schedule σ^* is optimal since for small value of ε and large T both machine A and B have equal workload. The ratio $\frac{C_{\max}\left(\sigma^{(1)}\right)}{C_{\max}(\sigma^*)}$ is 2 when $T \to \infty$ and $\varepsilon \to 0$. Therefore, the worst case bound of 2 is tight. ■

Cass Business School
CITY UNIVERSITY LONDON

FACULTY OF ACTUARIAL SCIENCE AND INSURANCE

Actuarial Research Papers since 2001

Report Number	Date	Publication Title	Author
135.	February 2001.	On the Forecasting of Mortality Reduction Factors. ISBN 1 901615 56 1	Steven Haberman Arthur E. Renshaw
136.	February 2001.	Multiple State Models, Simulation and Insurer Insolvency. ISBN 1 901615 57 X	Steve Haberman Zoltan Butt Ben Rickayzen
137.	September 2001	A Cash-Flow Approach to Pension Funding. ISBN 1 901615 58 8	M. Zaki Khorasanee
138.	November 2001	Addendum to "Analytic and Bootstrap Estimates of Prediction Errors in Claims Reserving". ISBN 1 901615 59 6	Peter D. England
139.	November 2001	A Bayesian Generalised Linear Model for the Bornhuetter-Ferguson Method of Claims Reserving. ISBN 1 901615 62 6	Richard J. Verrall
140.	January 2002	Lee-Carter Mortality Forecasting, a Parallel GLM Approach, England and Wales Mortality Projections. ISBN 1 901615 63 4	Arthur E.Renshaw Steven Haberman.
141.	January 2002	Valuation of Guaranteed Annuity Conversion Options. ISBN 1 901615 64 2	Laura Ballotta Steven Haberman
142.	April 2002	Application of Frailty-Based Mortality Models to Insurance Data. ISBN 1 901615 65 0	Zoltan Butt Steven Haberman
143.	Available 2003	Optimal Premium Pricing in Motor Insurance: A Discrete Approximation.	Russell J. Gerrard Celia Glass
144.	December 2002	The Neighbourhood Health Economy. A Systematic Approach to the Examination of Health and Social Risks at Neighbourhood Level. ISBN 1 901615 66 9	Les Mayhew
145.	January 2003	The Fair Valuation Problem of Guaranteed Annuity Options : The Stochastic Mortality Environment Case. ISBN 1 901615 67 7	Laura Ballotta Steven Haberman
146.	February 2003	Modelling and Valuation of Guarantees in With-Profit and Unitised With-Profit Life Insurance Contracts. ISBN 1 901615 68 5	Steven Haberman Laura Ballotta Nan Want
147.	March 2003.	Optimal Retention Levels, Given the Joint Survival of Cedent and Reinsurer. ISBN 1 901615 69 3	Z. G. Ignatov Z.G., V.Kaishev R.S. Krachunov
148.	March 2003.	Efficient Asset Valuation Methods for Pension Plans. ISBN 1 901615707	M. Iqbal Owadally
149.	March 2003	Pension Funding and the Actuarial Assumption Concerning Investment Returns. ISBN 1 901615 71 5	M. Iqbal Owadally

Report Number	Date	Publication Title	Author
150.	Available August 2004	Finite time Ruin Probabilities for Continuous Claims Severities	D. Dimitrova Z. Ignatov V. Kaishev
151.	August 2004	Application of Stochastic Methods in the Valuation of Social Security Pension Schemes. ISBN 1 901615 72 3	Subramaniam Iyer
152.	October 2003.	Guarantees in with-profit and Unitized with profit Life Insurance Contracts; Fair Valuation Problem in Presence of the Default Option[1]. ISBN 1-901615-73-1	Laura Ballotta Steven Haberman Nan Wang
153.	December 2003	Lee-Carter Mortality Forecasting Incorporating Bivariate Time Series. ISBN 1-901615-75-8	Arthur E. Renshaw Steven Haberman
154.	March 2004.	Operational Risk with Bayesian Networks Modelling. ISBN 1-901615-76-6	Robert G. Cowell Yuen Y, Khuen Richard J. Verrall
155.	March 2004.	The Income Drawdown Option: Quadratic Loss. ISBN 1 901615 7 4	Russell Gerrard Steven Haberman Bjorn Hojgarrd Elena Vigna
156.	April 2004	An International Comparison of Long-Term Care Arrangements. An Investigation into the Equity, Efficiency and sustainability of the Long-Term Care Systems in Germany, Japan, Sweden, the United Kingdom and the United States. ISBN 1 901615 78 2	Martin Karlsson Les Mayhew Robert Plumb Ben D. Rickayzen
157.	June 2004	Alternative Framework for the Fair Valuation of Participating Life Insurance Contracts. ISBN 1 901615-79-0	Laura Ballotta
158.	July 2004.	An Asset Allocation Strategy for a Risk Reserve considering both Risk and Profit. ISBN 1 901615-80-4	Nan Wang
159.	December 2004	Upper and Lower Bounds of Present Value Distributions of Life Insurance Contracts with Disability Related Benefits. ISBN 1 901615-83-9	Jaap Spreeuw
160.	January 2005	Mortality Reduction Factors Incorporating Cohort Effects. ISBN 1 90161584 7	Arthur E. Renshaw Steven Haberman
161.	February 2005	The Management of De-Cumulation Risks in a Defined Contribution Environment. ISBN 1 901615 85 5.	Russell J. Gerrard Steven Haberman Elena Vigna
162.	May 2005	The IASB Insurance Project for Life Insurance Contracts: Impart on Reserving Methods and Solvency Requirements. ISBN 1-901615 86 3.	Laura Ballotta Giorgia Esposito Steven Haberman
163.	September 2005	Asymptotic and Numerical Analysis of the Optimal Investment Strategy for an Insurer. ISBN 1-901615-88-X	Paul Emms Steven Haberman
164.	October 2005.	Modelling the Joint Distribution of Competing Risks Survival Times using Copula Functions. I SBN 1-901615-89-8	Vladimir Kaishev Dimitrina S, Dimitrova Steven Haberman
165.	November 2005.	Excess of Loss Reinsurance Under Joint Survival Optimality. ISBN1-901615-90-1	Vladimir K. Kaishev Dimitrina S. Dimitrova

Report Number	Date	Publication Title	Author
166.	November 2005.	Lee-Carter Goes Risk-Neutral. An Application to the Italian Annuity Market. ISBN 1-901615-91-X	Enrico Biffis Michel Denuit
167.	November 2005	Lee-Carter Mortality Forecasting: Application to the Italian Population. ISBN 1-901615-93-6	Steven Haberman Maria Russolillo
168.	February 2006	The Probationary Period as a Screening Device: Competitive Markets. ISBN 1-901615-95-2	Jaap Spreeuw Martin Karlsson
169.	February 2006	Types of Dependence and Time-dependent Association between Two Lifetimes in Single Parameter Copula Models. ISBN 1-901615-96-0	Jaap Spreeuw
170.	April 2006	Modelling Stochastic Bivariate Mortality ISBN 1-901615-97-9	Elisa Luciano Jaap Spreeuw Elena Vigna.
171.	February 2006	Optimal Strategies for Pricing General Insurance. ISBN 1901615-98-7	Paul Emms Steve Haberman Irene Savoulli
172.	February 2006	Dynamic Pricing of General Insurance in a Competitive Market. ISBN1-901615-99-5	Paul Emms
173.	February 2006	Pricing General Insurance with Constraints. ISBN 1-905752-00-8	Paul Emms
174.	May 2006	Investigating the Market Potential for Customised Long Term Care Insurance Products. ISBN 1-905752-01-6	Martin Karlsson Les Mayhew Ben Rickayzen
175.	December 2006	Pricing and Capital Requirements for With Profit Contracts: Modelling Considerations. ISBN 1-905752-04-0	Laura Ballotta
176.	December 2006	Modelling the Fair Value of Annuities Contracts: The Impact of Interest Rate Risk and Mortality Risk. ISBN 1-905752-05-9	Laura Ballotta Giorgia Esposito Steven Haberman
177.	December 2006	Using Queuing Theory to Analyse Completion Times in Accident and Emergency Departments in the Light of the Government 4-hour Target. ISBN 978-1-905752-06-5	Les Mayhew David Smith
178.	April 2007	In Sickness and in Health? Dynamics of Health and Cohabitation in the United Kingdom. ISBN 978-1-905752-07-2	Martin Karlsson Les Mayhew Ben Rickayzen
179.	April 2007	GeD Spline Estimation of Multivariate Archimedean Copulas. ISBN 978-1-905752-08-9	Dimitrina Dimitrova Vladimir Kaishev Spiridon Penev
180.	May 2007	An Analysis of Disability-linked Annuities. ISBN 978-1-905752-09-6	Ben Rickayzen
181.	May 2007	On Simulation-based Approaches to Risk Measurement in Mortality with Specific Reference to Poisson lee-Carter Modelling. ISBN 978-1-905752-10-2	Arthur Renshaw Steven Haberman

Report Number	Date	Publication Title	Author
182.	July 2007	High Dimensional Modelling and Simulation with Asymmetric Normal Mixtures. ISBN 978-1-905752-11-9	Andreas Tsanakas Andrew Smith
183.	August 2007	Intertemporal Dynamic Asset Allocation for Defined Contribution Pension Schemes. ISBN 978-1-905752-12-6	David Blake Douglas Wright Yumeng Zhang
184.	October 2007	To split or not to split: Capital allocation with convex risk measures. ISBN 978-1-905752-13-3	Andreas Tsanakas
185.	April 2008	On Some Mixture Distribution and Their Extreme Behaviour. ISBN 978-1-905752-14-0	Vladimir Kaishev Jae Hoon Jho
186.	March 2008	Optimal Funding and Investment Strategies in Defined Contribution Pension Plans under Epstein-Zin Utility. ISBN 978-1-905752-15-7	David Blake Douglas Wright Yumeng Zhang
187.	May 2008	Mortality Risk and the Valuation of Annuities with Guaranteed Minimum Death Benefit Options: Application to the Italian Population. ISBN 978-1-905752-16-4	Steven Haberman Gabriella Piscopo

Statistical Research Papers

1.	December 1995.	Some Results on the Derivatives of Matrix Functions. ISBN 1 874 770 83 2	P. Sebastiani
2.	March 1996	Coherent Criteria for Optimal Experimental Design. ISBN 1 874 770 86 7	A.P. Dawid P. Sebastiani
3.	March 1996	Maximum Entropy Sampling and Optimal Bayesian Experimental Design. ISBN 1 874 770 87 5	P. Sebastiani H.P. Wynn
4.	May 1996	A Note on D-optimal Designs for a Logistic Regression Model. ISBN 1 874 770 92 1	P. Sebastiani R. Settimi
5.	August 1996	First-order Optimal Designs for Non Linear Models. ISBN 1 874 770 95 6	P. Sebastiani R. Settimi
6.	September 1996	A Business Process Approach to Maintenance: Measurement, Decision and Control. ISBN 1 874 770 96 4	Martin J. Newby
7.	September 1996.	Moments and Generating Functions for the Absorption Distribution and its Negative Binomial Analogue. ISBN 1 874 770 97 2	Martin J. Newby
8.	November 1996.	Mixture Reduction via Predictive Scores. ISBN 1 874 770 98 0	Robert G. Cowell.
9.	March 1997.	Robust Parameter Learning in Bayesian Networks with Missing Data. ISBN 1 901615 00 6	P.Sebastiani M. Ramoni
10.	March 1997.	Guidelines for Corrective Replacement Based on Low Stochastic Structure Assumptions. ISBN 1 901615 01 4.	M.J. Newby F.P.A. Coolen
11.	March 1997	Approximations for the Absorption Distribution and its Negative Binomial Analogue. ISBN 1 901615 02 2	Martin J. Newby
12.	June 1997	The Use of Exogenous Knowledge to Learn Bayesian Networks from Incomplete Databases. ISBN 1 901615 10 3	M. Ramoni P. Sebastiani

Report Number	Date	Publication Title	Author
13.	June 1997	Learning Bayesian Networks from Incomplete Databases. ISBN 1 901615 11 1	M. Ramoni P.Sebastiani
14.	June 1997	Risk Based Optimal Designs. ISBN 1 901615 13 8	P.Sebastiani H.P. Wynn
15.	June 1997.	Sampling without Replacement in Junction Trees. ISBN 1 901615 14 6	Robert G. Cowell
16.	July 1997	Optimal Overhaul Intervals with Imperfect Inspection and Repair. ISBN 1 901615 15 4	Richard A. Dagg Martin J. Newby
17.	October 1997	Bayesian Experimental Design and Shannon Information. ISBN 1 901615 17 0	P. Sebastiani. H.P. Wynn
18.	November 1997.	A Characterisation of Phase Type Distributions. ISBN 1 901615 18 9	Linda C. Wolstenholme
19.	December 1997	A Comparison of Models for Probability of Detection (POD) Curves. ISBN 1 901615 21 9	Wolstenholme L.C
20.	February 1999.	Parameter Learning from Incomplete Data Using Maximum Entropy I: Principles. ISBN 1 901615 37 5	Robert G. Cowell
21.	November 1999	Parameter Learning from Incomplete Data Using Maximum Entropy II: Application to Bayesian Networks. ISBN 1 901615 40 5	Robert G. Cowell
22.	March 2001	FINEX : Forensic Identification by Network Expert Systems. ISBN 1 901615 60X	Robert G.Cowell
23.	March 2001.	Wren Learning Bayesian Networks from Data, using Conditional Independence Tests is Equivalant to a Scoring Metric ISBN 1 901615 61 8	Robert G Cowell
24.	August 2004	Automatic, Computer Aided Geometric Design of Free-Knot, Regression Splines. ISBN 1-901615-81-2	Vladimir K Kaishev, Dimitrina S.Dimitrova, Steven Haberman Richard J. Verrall
25.	December 2004	Identification and Separation of DNA Mixtures Using Peak Area Information. ISBN 1-901615-82-0	R.G.Cowell S.L.Lauritzen J Mortera,
26.	November 2005.	The Quest for a Donor : Probability Based Methods Offer Help. ISBN 1-90161592-8	P.F.Mostad T. Egeland., R.G. Cowell V. Bosnes Ø. Braaten
27.	February 2006	Identification and Separation of DNA Mixtures Using Peak Area Information. (Updated Version of Research Report Number 25). ISBN 1-901615-94-4	R.G.Cowell S.L.Lauritzen J Mortera,
28.	October 2006	Geometrically Designed, Variable Knot Regression Splines : Asymptotics and Inference. ISBN 1-905752-02-4	Vladimir K Kaishev Dimitrina S.Dimitrova Steven Haberman Richard J. Verrall

Faculty of Actuarial Science and Insurance

Actuarial Research Club

The support of the corporate members

- CGNU Assurance
- English Matthews Brockman
- Government Actuary's Department

is gratefully acknowledged.

ISBN 978-1-905752-18-8