

# Decentralization Cost in Supply Chain Jobshop Scheduling with Minimum Flowtime Objective

Yossi Bukchin\*      Eran Hanany†

January 2011

## Abstract

Scheduling problems naturally arise in supply chains, ranging from manufacturing and business levels in a single firm through multi-organizational processes. In these environments, jobs are served by various resources. As scheduling choices are often decentralized, an inferior system solution may occur. In this paper we investigate a decentralized jobshop system, where jobs are processed utilizing two resources, each of which minimizing their own flowtime objective. Analyzing the system as a non-cooperative game, we investigate the decentralization cost (DC), the ratio between the best Nash equilibrium and the centralized solution. Inefficiency can occur for small as well as large instances. Lower bounds on the maximal DC are provided for the jobshop and flowshop environments, leading to significant values. We conclude that managers within such decentralized systems should implement strategies that overcome the inherent inefficiency. A simple mechanism is proposed to reduce the DC value, and its useful effectiveness is analyzed both analytically and empirically.

## 1 Introduction

Scheduling problems deal with allocation of scarce resources to tasks over time while optimizing one or more objectives (Pinedo 2002). Such problems can be found in manufacturing and service environments. The traditional scheduling literature considers a centralized system with a single decision maker (DM) and suggests optimization models and heuristics minimizing objectives such as makespan, flowtime, tardiness and others. In reality, however, quite often several DMs are

---

\*Department of Industrial Engineering, Tel Aviv University, Tel Aviv 69978, Israel. Email: bukchin@eng.tau.ac.il

†Department of Industrial Engineering, Tel Aviv University, Tel Aviv 69978, Israel. Email: hananye@post.tau.ac.il

involved in the scheduling decision process, each aiming at maximizing their own utility. Decisions may be taken by local DMs since a central DM (system manager) either (1) does not exist; (2) does not have sufficient information; (3) is not able to provide an optimal solution due to the problem complexity; and/or (4) does not see the necessity in enforcing a centralized solution.

A decentralized process where decisions made by some or all DMs are influenced by those made by other DMs can be modeled using game theory approaches. The research to date combining multiple DMs and scheduling decisions considers single or parallel (identical, related or unrelated) machine/resources, where DMs could be either jobs or machines (this literature is further discussed later in the Introduction). However, in many real world situations the scheduling problem is more general. In this paper we consider a decentralized setting of multiple DMs in a jobshop environment, whereby each DM manages a different resource serving multiple jobs, and each job passes along a predetermined route consisting of several resources. In doing so we provide the first model and analysis of decentralized jobshop settings. Decentralized jobshop systems can be found in both manufacturing and service real world situations. In the former, jobs are processed in different departments or on different machines, where each department manager is responsible to the scheduling of jobs on the department resources, while optimizing their own performance measures. Such situations may also occur on the operational level of non-manufacturing environments, where tasks are performed by different functions of the organization. One of the main causes for the emergence of coordination problems in such settings is the matrix structure that many firms adopt to improve their agility and flexibility, as compared to the traditional, functional structure. In such organizations, project or product managers utilize the services of functional departments (R&D, marketing and sales, human resource, finance and accounting, distribution, etc.). Each functional department manager handles a set of jobs sent by the project managers, and may schedule these jobs to maximize their own benefit. Consequently the balance between managerial power of functional department managers versus project managers depends on the type of matrix structuring the organization (a weak, functional matrix versus a strong, project oriented matrix). A decentralized jobshop setting may also describe a supply network (Harland et al. 2001), where the resources are different companies involved in performing multiple projects. Although such an environment is typically decentralized by ownership, the companies may be interested in analyzing the performance of the distributed system versus a centralized one. Poor performance may lead to collaboration of the parties in order to coordinate the system.

To address these issues we take a non-cooperative game theoretic approach and analyze a

decentralized setting where the centralized objective is total flowtime minimization. The scheduling decisions on each resource are taken by the corresponding resource manager. The objective function of each resource manager is the same type as the objective function of the system, namely, the flowtime objective on their own resource. The flowtime for each resource is defined by the sum of flowtimes over all jobs on this resource. The flowtime of a job on a resource is defined by the difference between the completion time of the job on this resource and the ready time of the job (also typically termed release time), i.e., the time the job was available at the resource site. A static problem is considered, in which all jobs are simultaneously ready at their respective initial machine at time zero. We also consider the special case of a flowshop setting, in which all jobs have the same route. All the parameters of the model are commonly known to the DMs.

The objective of the paper is to analyze the *decentralization cost* (DC), i.e., the ratio between the *best* decentralized Nash equilibrium solution and the centralized optimal solution in the jobshop environment. Following this analysis we are able to draw managerial insights on the merit of intervention by a centralized DM, or the implementation of an improving mechanism. Our main conclusion is that although both the system manager and each of the resource managers aim in minimizing the same type of objective function, the DC can be significant. Therefore, managers within such decentralized systems may wish to implement strategies that overcome the inherent inefficiency. To this end, we suggest a simple improving mechanism, and show that it yields the centralized solution in some of the worst cases that may occur when no intervention is applied.

In the remainder of the Introduction we discuss the current literature on scheduling games. The first papers that combined game theory and scheduling were based on cooperative games, analyzing the possibility and benefit of cooperation among players competing on a common resource. Curiel et al. (1989) initiated this line of research by proposing the single machine sequencing game, in which several jobs, each belonging to a different agent, must be processed on a single machine. The authors suggest that an allocation rule induced by a cooperative game can allocate the cost saving that results from coalitions' decisions to move from the initial sequence to an optimal one. This model has been extended to consider ready times of the jobs (Hamers et al. 1995), due dates (Borm et al. 2002), multiple parallel identical machines (Hamers et al. 1999), more general interchanges (Slikker 2006) and outsourcing operations to a third party (Aydinliyim and Vairaktarakis 2010a). Other cooperative game scheduling models are surveyed in Borm et al. (2001), Curiel et al. (2002), Hall and Liu (2008) and Aydinliyim and Vairaktarakis (2010b).

There has also been a growing literature on non-cooperative games in scheduling settings, as

it is natural to assume that DMs may not cooperate with each other when making decisions. Koutsoupias and Papadimitriou (1999) and others analyzed scheduling environments in the spirit of congestion games (Rosenthal 1973), whereby each job is represented by a DM choosing a machine from a set of related parallel machines and all jobs are processed and released as a batch. Under the objective of minimum makespan and complete information they provided bounds for the *price of anarchy*, the ratio of the *worst* equilibrium to the optimal solution. Analogue bounds were provided by Christodoulou et al. (2004) and Immorlica et al. (2005) and others for sequencing settings with unrelated/related machines where jobs are released immediately upon processing completion. Bukchin and Hanany (2007) analyzed a different setting whereby each DM owns a set of jobs and may choose between processing on a common in-house resource or on a slower but uncapacitated subcontractor, i.e., a setting of related machines. Under the minimum individual and system total flowtime objective they investigated the DC, provided bounds on it, found empirically DC values of up to around 1.35 and proposed a coordinating mechanism based solely on scheduling decisions, i.e., with no monetary transfers between the players. Correa and Queyranne (2010) provide bounds for the price of anarchy under the weighted completion time objective in a setting where DMs, each having a single job, choose between related machines each only capable of processing a subset of jobs.

Nisan and Ronen (2001) initiated a line of research on mechanism design of centralized scheduling in settings where DMs have private information and the proposed mechanism uses money transfers between the DMs to induce truthfull information provision to the central manager. A mechanism design approach achieving truth revelation and/or non-costly money transfers to the central manager (i.e., money transfers that add up to zero) is not always possible in general (Jéhiel and Moldovanu 2001), specifically when the private information is multi-dimensional such as when DMs own more than one job (Hain and Mitra 2004 proposed a non-costly mechanism in a single machine setting with DMs each owning a single job). Environments in which agents compete on a common resource were also investigated via auction games, in which the resource owner allocates time slots to the agents based on their bidding (see e.g., Kutanoglu and Wu 1999, Wellman et al. 2001 and Reeves et al. 2005). For further discussion of literature on non-cooperative scheduling games see Heydenreich et al. (2007), Hall and Liu (2008) and Aydinliyim and Vairaktarakis (2010b).

The paper is organized as follows. In Section 2 we describe our model of decentralized jobshop scheduling, including several examples demonstrating the properties of Nash equilibrium outcomes

compared to centralized optimal outcomes. In Section 3 we analyze the DC for jobshop settings and provide a significant lower bound, suggesting that the DC should not be ignored. In Section 4 we undertake a similar analysis for flowshop settings, and show that flowshop restrictions do not significantly reduce DC problems. In light of these conclusions, in Section 5 we provide a simple mechanism towards solving the problems raised in previous sections. In Section 6 we complement the analytic analysis of previous sections with numerical analysis emphasizing our conclusions. Section 7 concludes.

## 2 Modelling decentralized jobshop scheduling

Consider  $J \geq 2$  jobs  $j \in \{1, \dots, J\}$  to be processed in serial on two machines  $m \in M = \{1, 2\}$ , with processing durations  $t(j, m) \geq 0$ . Each job may require its own different route and is available at the system at time 0 to be processed on the first machine. Note that for concreteness we refer throughout the paper to job processing on machines, but our analysis applies more generally also to service contexts, in which servers or capacitated resources operate instead of machines. We will refer to any job for which machine  $m$  is first in the job's processing order as a job belonging to machine  $m$  (or alternatively say that machine  $m$  owns the job). Denote by  $J_m$  the number of jobs belonging to machine  $m$ , thus  $\sum_{m \in M} J_m = J$ . Denote by  $p(j, m)$  the machine preceding  $m$  in job  $j$ 's processing order, with the convention that  $p(j, m) = 0$  when  $j$  belongs to  $m$ . Each machine  $m$  determines its own processing sequence permutation independently from the other machine. Denote by  $q(o, m) \in \{1, \dots, J\}$  the job sequenced to be processed in the  $o^{\text{th}}$  position on machine  $m$ , and correspondingly  $o_{j,m} \in \{1, \dots, J\}$  denotes the position in the sequence on this machine for job  $j$ , satisfying  $q(o_{j,m}, m) = j$ . Given these sequences, each job  $j$  starts its processing on machine  $m$  as soon as it arrives to the machine and its preceding job on this machine,  $q(o_{j,m} - 1, m)$ , completed its processing. Upon processing completion on machine  $m$ , the job immediately arrives to the next machine in its processing order or leaves the system. Therefore the starting time,  $s_q(j, m)$ , of job  $j$  on machine  $m$  in sequence  $q$  is defined recursively as

$$s_q(j, m) \equiv \max\{s_q(j, p(j, m)) + t(j, p(j, m)), s_q(q(o_{j,m} - 1, m), m) + t(q(o_{j,m} - 1, m), m)\}, \quad (1)$$

with initial conditions  $s_q(j, 0) = t(j, 0) = s_q(0, m) = t(0, m) = q(0, m) = 0$  for all  $j$  and  $m$ . The first term within this maximum is the completion time of job  $j$  on the machine preceding  $m$  in this job's processing order (equal to zero when  $m$  is the first machine) and the second term is the

completion time of the job preceding job  $j$  on machine  $m$  (equal to zero when  $j$  is the first job). Each machine  $m$  determines its sequence with the objective of minimizing its own total flowtime, i.e., the sum of completion times minus ready times on machine  $m$ ,

$$F_m^q \equiv \sum_{j=1}^J (s_q(j, m) + t(j, m)) - \sum_{j=1}^J (s_q(j, p(j, m)) + t(j, p(j, m))), \quad (2)$$

where each machine takes the other machine's sequence as given, thus making  $q$  a Nash equilibrium. It follows that equilibrium corresponds to simultaneous scheduling of all jobs on each machine, with interdependent ready times, to minimize flowtime. Efficiency is measured by the system flowtime,  $F^q \equiv F_1^q + F_2^q$ , thus an optimal solution is a two machine jobshop schedule minimizing total flowtime. The *decentralization cost*<sup>1</sup> (DC) (Bukchin and Hanany 2007) is the ratio between the minimal  $F^q$  among all Nash equilibria  $q$  and the minimal  $F^q$  among all possible sequences  $q$ .

To illustrate, consider two jobs belonging to different machines, i.e.,  $p(j, j) = 0$ ,  $p(j, 3 - j) = j$  and  $J_m = 1$  for each  $j, m$ . There are four possible sequences  $q$ , each corresponding to a pair of job permutations. Note however that the sequence defined by  $q(1, 1) = 2$ ,  $q(2, 1) = 1$ ,  $q(1, 2) = 1$  and  $q(2, 2) = 2$  is infeasible because each job waits to start its processing on the first machine after the completion of the other job, which is waiting on the other machine, leading to a deadlock. The remaining three sequences are feasible. Figure 1 depicts the flowtimes of each schedule and the Gantt chart of the equilibrium schedule for two instances of the problem, where the processing times are given in the two matrices on the left (the notation  $(1 - 1)$  in the title of the figure refers to  $(J_1, J_2)$ ). The rows and the columns in each matrix of flowtimes correspond to the vectors  $\{q(o, 1), 1 \leq o \leq J\}$  and  $\{q(o, 2), 1 \leq o \leq J\}$ , respectively, and each cell corresponds to the vector

---

<sup>1</sup>Similar concepts named *price of anarchy* and *price of stability* are commonly used in the computer science literature (see, e.g., Heydenreich et al. 2007 and Anshelevich et al. 2003), where the latter is the same as the DC, while the former refers to the worst Nash equilibrium rather than the best one as in the DC. For arguments justifying the DC definition see Bukchin and Hanany (2007).

$(F_1^q, F_2^q)$ .

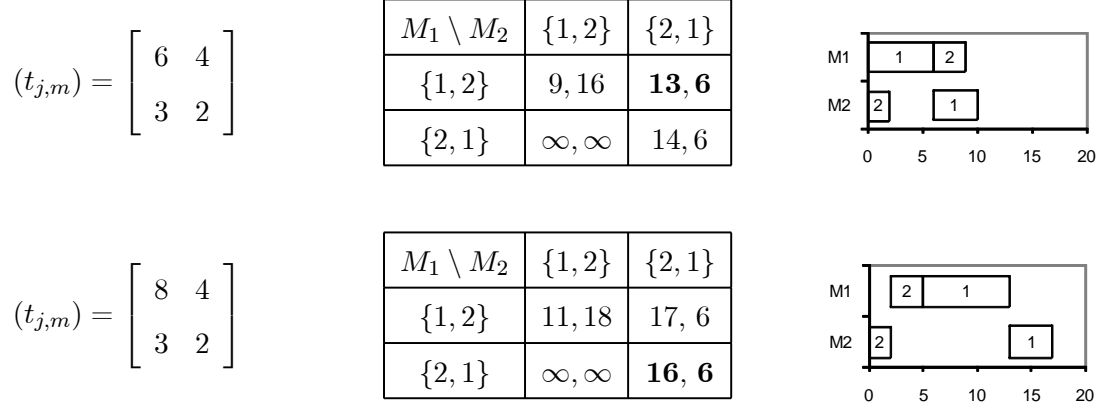


Figure 1: Two instances with two jobs (1 – 1)

In the first instance in Figure 1, each machine processes first its own job and only then the other machine's job, while in the second instance, machine 1 prefers to process first the other machine's job, only then to process its own job.

Note that for both instances, the equilibrium outcome, marked in the matrix of flowtimes with bold letters, is also the optimal schedule. This property is not special to the processing durations chosen in these examples, as shown in the next theorem.

**Theorem 1** *For all jobshop settings with two jobs, where  $J_m = 1$  for each  $m$ , the DC equals 1.*

**Proof.** Note that the flowtime  $F_m^q$  of each of the feasible sequences can be computed for each machine as follows.

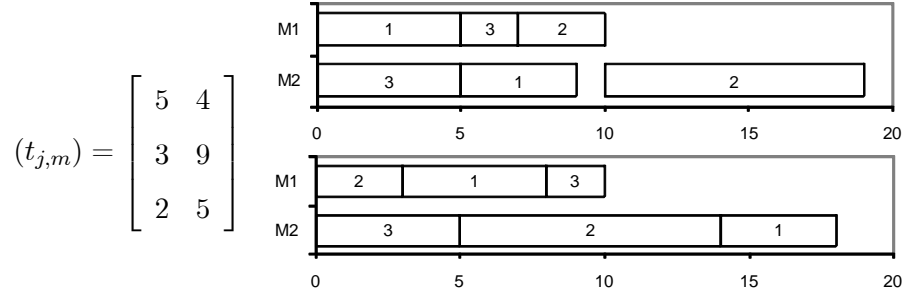
$M_1 \setminus M_2$	$\{1, 2\}$	$\{2, 1\}$
$\{1, 2\}$	$t(1, 1) + t(2, 1),$ $t(1, 1) + 2t(1, 2) + t(2, 2)$	$t(1, 1) + \max\{t(1, 1), t(2, 2)\} + t(2, 1) - t(2, 2),$ $t(2, 2) + \max\{t(2, 2), t(1, 1)\} + t(1, 2) - t(1, 1)$
$\{2, 1\}$	$\infty, \infty$	$t(2, 2) + 2t(2, 1) + t(1, 1),$ $t(2, 2) + t(1, 2)$

We first show that if  $q^* = \{\{1, 2\}, \{2, 1\}\}$  is optimal, then it is also a Nash equilibrium. Let  $q_1 = \{\{2, 1\}, \{2, 1\}\}$  and  $q_2 = \{\{1, 2\}, \{1, 2\}\}$ . Therefore  $F_1^{q_2} - F_1^{q^*} = t(2, 2) - \max\{t(1, 1), t(2, 2)\} \leq 0$  and  $F_2^{q_1} - F_2^{q^*} = t(1, 1) - \max\{t(1, 1), t(2, 2)\} \leq 0$ . Thus  $F_1^{q_1} - F_1^{q^*} \geq (F_1^{q_1} + F_2^{q_1}) - (F_1^{q^*} + F_2^{q^*})$

and  $F_2^{q_2} - F_2^{q^*} \geq (F_1^{q_2} + F_2^{q_2}) - (F_1^{q^*} + F_2^{q^*})$ . By optimality of  $q^*$ , the right-hand side of each of these inequalities is non-negative, thus  $F_1^{q_1} - F_1^{q^*} \geq 0$  and  $F_2^{q_2} - F_2^{q^*} \geq 0$ . Therefore  $q^*$  is a Nash equilibrium.

We now show that if  $\hat{q} = \{\{2, 1\}, \{2, 1\}\}$  is optimal, then it is also a Nash equilibrium. Let  $\bar{q} = \{\{1, 2\}, \{2, 1\}\}$ . By optimality of  $\hat{q}$ ,  $F_2^{\bar{q}} - F_2^{\hat{q}} \geq F_1^{\hat{q}} - F_1^{\bar{q}}$ . Assume that  $\hat{q}$  is not a Nash equilibrium. Since  $\{\{2, 1\}, \{1, 2\}\}$  is infeasible, for  $\hat{q}$  not to be a Nash equilibrium it must be that  $F_1^{\hat{q}} - F_1^{\bar{q}} > 0$ . Combining the above,  $F_2^{\bar{q}} - F_2^{\hat{q}} > 0$ . Since  $F_2^{\bar{q}} - F_2^{\hat{q}} = \max\{t(1, 1), t(2, 2)\} - t(1, 1)$ , it follows that  $t(2, 2) > t(1, 1)$ . Since  $F_1^{\hat{q}} - F_1^{\bar{q}} = t(2, 1) + 2t(2, 2) - \max\{t(1, 1), t(2, 2)\}$ ,  $F_2^{\bar{q}} - F_2^{\hat{q}} \geq F_1^{\hat{q}} - F_1^{\bar{q}}$  implies  $-t(1, 1) \geq t(2, 1)$ , contradicting positive processing durations on machine 1. Therefore  $\hat{q}$  must be an equilibrium. A similar argument shows that if  $\{\{1, 2\}, \{1, 2\}\}$  is optimal then it is also a Nash equilibrium. ■

Note Theorem 1 does not hold for a flowshop setting with two jobs. Also, if we add another single job to the setting of the theorem, the argument in the proof fails, as demonstrated in the example in Figure 2. Here machine 1 owns jobs 1, 2 and machine 2 owns job 3. The top and bottom Gantt charts show the unique optimal schedule and the unique equilibrium schedule, respectively (the corresponding flowtimes are marked in the matrix below the charts with bold letters). The DC obtained for this example is equal to 1.2.



$M_1 \setminus M_2$	$\{1, 2, 3\}$	$\{1, 3, 2\}$	$\{2, 1, 3\}$	$\{3, 1, 2\}$	$\{2, 3, 1\}$	$\{3, 2, 1\}$
$\{1, 2, 3\}$	15, 37	15, 33	15, 51	18, 19	15, 52	18, 30
$\{1, 3, 2\}$	$\infty, \infty$	26, 27	$\infty, \infty$	<b>17, 18</b>	$\infty, \infty$	17, 32
$\{2, 1, 3\}$	13, 48	13, 44	13, 38	16, 27	13, 39	<b>16, 26</b>
$\{3, 1, 2\}$	$\infty, \infty$	$\infty, \infty$	$\infty, \infty$	29, 19	$\infty, \infty$	29, 30
$\{2, 3, 1\}$	$\infty, \infty$	$\infty, \infty$	$\infty, \infty$	17, 31	29, 30	17, 22
$\{3, 2, 1\}$	$\infty, \infty$	$\infty, \infty$	$\infty, \infty$	27, 27	$\infty, \infty$	27, 22

Figure 2: Instance with three jobs (2 – 1) and  $DC = \frac{42}{35} = 1.2$  .

### 3 Analysis of the DC in jobshop settings

In this section we present a general structure of jobs that leads to large values of DC in jobshop settings. This analysis provides a lower bound on the maximal DC. The general structure involves jobs with small differences in processing durations (i.e., the variance in processing durations is small relative to their mean) such that the processing duration is smaller on the initial machine than on the later machine, possibly justified due to some setup or handling time. Under the additional assumption that the processing durations are identical on the initial machine and identical on the later machine, the following theorem provides the resulting limiting lower bound on the maximal DC.

**Theorem 2** *A lower bound on the maximal DC among all possible instances with  $J$  jobs approaches*

$$\frac{3\lfloor \frac{J}{2} \rfloor + 1}{2(\lfloor \frac{J}{2} \rfloor + 1)},$$

*and is achieved in a jobshop setting in which each machine owns half of the jobs, i.e.,  $J_1 = J - \lfloor \frac{J}{2} \rfloor$ ,*

$J_2 = \lfloor \frac{J}{2} \rfloor$ , and the processing durations are slightly smaller on the initial machine than on the later machine but are otherwise identical, i.e., for all  $j$  and  $m$  and some  $c > 0$  and  $\varepsilon \rightarrow 0^+$ ,  $t(j, m) = c - \varepsilon$  if  $m$  is the initial machine for job  $j$ , otherwise  $t(j, m) = c$ .

**Proof.** Without loss of generality we can index the jobs in an ascending order so that machine 1 owns the lower half of the jobs (plus one job when  $J$  is odd) and the upper half belongs to machine 2, i.e.,  $p(j, 1) = 0, p(j, 2) = 1$  for  $1 \leq j \leq J_1$  and  $p(j, 2) = 0, p(j, 1) = 2$  for  $J_1 + 1 \leq j \leq J$ , where  $J_1 = J - J_2 = J - \lfloor \frac{J}{2} \rfloor$ . Consider any sequence  $q^E$ , referred to as type  $E$  sequence, for which each machine processes its own jobs first and only then processes all the remaining jobs, i.e.,  $1 \leq q(o, 1) \leq J_1$  for  $1 \leq o \leq J_1$ ,  $J_1 + 1 \leq q(o, 1) \leq J$  for  $J_1 + 1 \leq o \leq J$ ,  $J_1 + 1 \leq q(o, 2) \leq J$  for  $1 \leq o \leq J_2$  and  $1 \leq q(o, 2) \leq J_1$  for  $J_2 + 1 \leq o \leq J$ . Type  $E$  sequence is illustrated in Figure 3 when  $J$  is even.

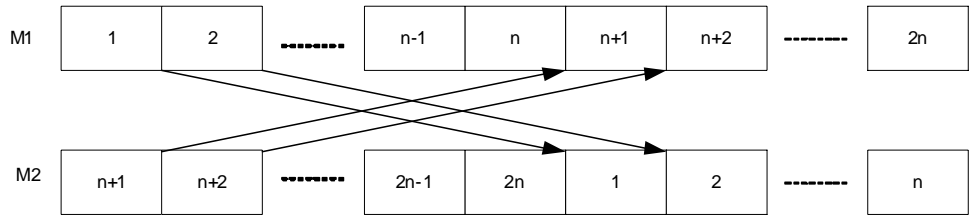


Figure 3: Type  $E$  sequence

We first show that any such sequence is a Nash equilibrium. To see this note that since processing durations are identical on the initial machine, when each machine completes processing all of its jobs, all other jobs will already have arrived from the other machine, thus all jobs are processed with no idle times. Moreover, since the sum of ready times, i.e., the second term in each machine's total flowtime (2), is determined by the other machine, this term is constant with respect to all possible sequences this machine can choose. Thus minimizing the machine's total flowtime is equivalent to minimizing the sum of completion times. Since this objective is achieved on a single machine by any SPT sequencing with no idle times, type  $E$  sequence is a best response for each machine, thus it is a Nash equilibrium.

Observe now that type  $E$  sequences are the only possible Nash equilibria. To see this note that the processing duration of each job is slightly smaller on the machine to which it belongs than on the later machine. Consider a sequence  $q$  which is not of type  $E$ . With a simple pairwise exchange of jobs on some machine we can show that such a sequence can be improved upon, hence cannot

be a best response. In particular, there must exist a pair of adjacent jobs in  $q$ , say  $j_2$  followed by  $j_1$ , on some machine  $\bar{m}$ , such that  $\bar{m}$  owns  $j_1$  and the other machine owns  $j_2$ . Moreover, there is no idle time between the pair of jobs on machine  $\bar{m}$  because  $j_1$  has ready time of 0. Consider a sequence  $q'$  that is identical to  $q$  except that  $j_1$  and  $j_2$  are swapped on  $\bar{m}$ . Since  $j_1$  has ready time 0 and  $j_2$  is delayed when going from  $q$  to  $q'$ , the swap does not cause additional idle times, thus there is no change in the total flowtime of  $\bar{m}$  due to any of the other jobs. However, since the processing duration of  $j_1$  on  $\bar{m}$  is slightly smaller than that of  $j_2$ , the swap reduces the total flowtime of  $\bar{m}$  by the difference in durations between the pair of jobs. Therefore sequence  $q$  is not a Nash equilibrium, thus showing that only type  $E$  sequences are equilibria.

We now show optimality for the system of any sequence  $q^O$ , referred to as type  $O$  sequence, for which the machines coordinate by simultaneously processing a sequence of pairs of jobs, each job in the pair belongs to a different machine (when  $J$  is odd, an additional job is processed after the last pair). For each such pair of jobs, each machine first processes its own job, then each job immediately starts processing on the other machine in order to leave the system as soon as possible. More specifically,  $q^O$  satisfies  $1 \leq q(o, 1) \leq J_1$ ,  $J_1 + 1 \leq q(o, 2) \leq J$  and  $q(o + 1, m) = q(o, 3 - m)$  for  $o = 1, 3, \dots, 2J_2 - 1$  and each  $m$ , and additionally  $1 \leq q(J, 1) \leq J_1$  and  $q(J, 2) = q(J, 1)$  when  $J$  is odd. Type  $O$  sequence is illustrated in Figure 4 when  $J$  is even.

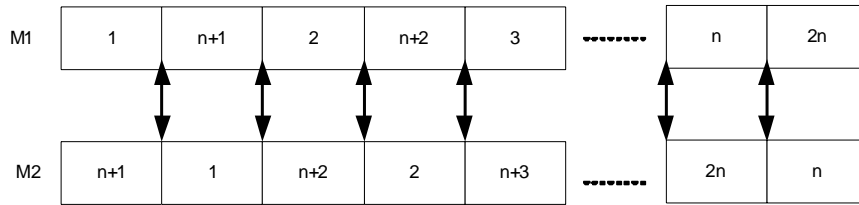


Figure 4: Type  $O$  sequence

Note that due to the identical processing durations for all jobs on the initial machine and on the later machine, a type  $O$  sequence utilizes the machines in the most efficient way possible. This is because the system completion time of any job (except for the last job when  $J$  is odd) exactly equals the lower bound defined as half the sum of processing durations over both machines and over all jobs that leave the system before and including this job. This immediately implies that a type  $O$  sequence is optimal for the system, because any other sequence may have additional idle times or additional terms added to a job's completion time due to processing durations of jobs that leave the system after this job (when  $J$  is odd the last job is processed on each machine while the

other machine is idle, however this schedule cannot be improved).

It remains to compute the limiting DC for this jobshop setting. According to a type  $E$  sequence, the system completion times in  $c$  multiples are almost  $J_2 + 1, J_2 + 2, \dots, J$  for machine 1's jobs and  $J_1 + 1, J_1 + 2, \dots, J$  for machine 2's jobs. According to a type  $O$  sequence, the system completion times in  $c$  multiples are almost  $2, 4, \dots, 2J_1$  for machine 1's jobs and  $2, 4, \dots, 2J_2$  for machine 2's jobs. It follows that the limiting DC is equal to

$$\frac{\frac{1}{2}J_1(J_2 + 1 + J) + \frac{1}{2}J_2(J_1 + 1 + J)}{\frac{1}{2}J_1(2 + 2J_1) + \frac{1}{2}J_2(2 + 2J_2)}.$$

The proof is established because this expression simplifies to  $\frac{3J_2+1}{2(J_2+1)}$  both when  $J_1 = J_2$  and when  $J_1 = J_2 + 1$ . ■

The value of the lower bound increases with the number jobs, attains relatively high values even for a small number of jobs and asymptotes to an upper bound of 1.5, as illustrated in Figure 5.

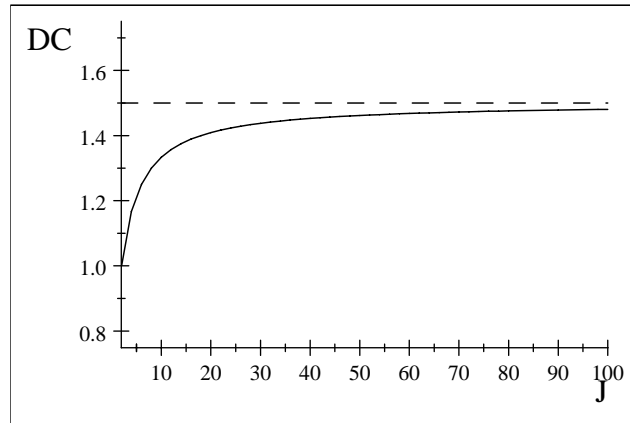


Figure 5: Lower bound on the maximal DC as a function of the number of jobs  $J$

One can show using simulation that similar results to that of Theorem 2 are obtained when the processing durations are not identical but are instead distributed with some variance while still being smaller on the initial machine than on the later machine.

## 4 Analysis of the DC in flowshop settings

For a flowshop setting, the machine processing order is identical for all jobs and can be assumed, without loss of generality, to be machine 1 followed by machine 2, i.e.,  $p(j, 1) = 0$  and  $p(j, 2) = 1$  for all jobs  $j$ . In this case, there is asymmetry in the sense that the flowtime of machine 2, which

depends on its ready times, is affected by the sequencing decision of machine 1, but not vice versa. Therefore an equilibrium outcome can be found using a greedy algorithm, in which an optimal sequence is first found for machine 1 irrespective of machine 2, and then an optimal sequence is computed for machine 2 given the ready times determined by the sequence on machine 1. Note that the second part of this algorithm is a hard problem, and that finding the optimal centralized sequence is hard as well (Lenstra et al. 1977). The expressions for the starting time  $s_q(j, m)$  of job  $j$  on machine  $m$  and machine  $m$ 's total flowtime, given by (1) and (2), simplify to

$$s_q(j, m) = \max\{s_q(j, m-1) + t(j, m-1), s_q(q(o_{j,m}-1, m), m) + t(q(o_{j,m}-1, m), m)\} \quad (3)$$

and

$$F_m^q = \sum_{j=1}^J (s_q(j, m) + t(j, m)) - \sum_{j=1}^J (s_q(j, m-1) + t(j, m-1)). \quad (4)$$

We now formulate a lower bound on the maximal DC.

**Theorem 3** *A lower bound on the maximal DC among all possible instances of  $J$  jobs in a flowshop setting approaches*

$$\max_{2 \leq k \leq J} \frac{\frac{1}{4}J(4 + 2J + 3k - k^2) + \sum_{i=J-k+2}^J \frac{1}{i-1} J^2 (1 + \frac{1}{2}(J-i))}{\frac{1}{2}J(J+3) + \sum_{i=J-k+2}^J \frac{1}{i-1} \frac{1}{2} J(J-i+1)(J-i+2)},$$

and is achieved for some  $2 \leq k \leq J$  with the processing durations  $t(j, 1) = 1 - j \cdot \varepsilon$  for all  $j$  and  $\varepsilon \rightarrow 0^+$ , and  $t(j, 2) = 1$  for  $1 \leq j \leq J - k + 1$  and  $t(j, 2) = 1 + \sum_{i=J-k+2}^j \frac{J}{i-1}$  for  $J - k + 2 \leq j \leq J$ .

**Proof.** Machine 1 strictly prefers an SPT sequencing of all jobs to minimize its flowtime,  $F_1^q$ , leading to the sequence  $\bar{q}$  defined by

$$\bar{q}(o, 1) = J - o + 1 \text{ for } 1 \leq o \leq J.$$

For simplicity of presentation we establish the remainder of the proof with  $\varepsilon = 0$ , as similar arguments are valid when  $\varepsilon \rightarrow 0^+$  (the only role of  $\varepsilon > 0$  is to eliminate Nash equilibria that are different from those obtained below, with almost no change in the system flowtime). Note that for machine 1 the starting time is  $s_{\bar{q}}(j, 1) = J - j$  for all  $1 \leq j \leq J$ . Taking this sequence imposed by machine 1 as given, we would like to obtain an optimal sequence for machine 2. To this end we augment  $\bar{q}$  with  $J$  particular sequences for this machine, denoted  $\bar{q}^{j'}$  for  $1 \leq j' \leq J$ , such that  $\bar{q}^{j'}$  starts with job  $j'$  and all other jobs follow SPT. We will prove for each  $j'$  that  $\bar{q}^{j'}$  is optimal for

machine 2 among all sequences starting with job  $j'$ . To prove this, it will be sufficient to show that all jobs after  $j'$  are scheduled with no idle times. This is true because the sum of ready times on machine 2,  $\sum_{j=1}^J (s_q(j, 1) + t(j, 1))$ , is the same for all sequences  $q$ , so minimizing machine 2's total flowtime,  $F_2^q$  (see eq. (4)), is equivalent to minimizing the sum of completion times on machine 2,  $\sum_{j=1}^J (s_q(j, 2) + t(j, 2))$ . Since all sequences starting with job  $j'$  on machine 2 have the same  $s_q(j', 2)$ , SPT sequencing of all other jobs minimizes the sum of completion times on machine 2. Consider  $\bar{q}^{j'}$  defined for  $1 \leq j' \leq J - k + 1$  by

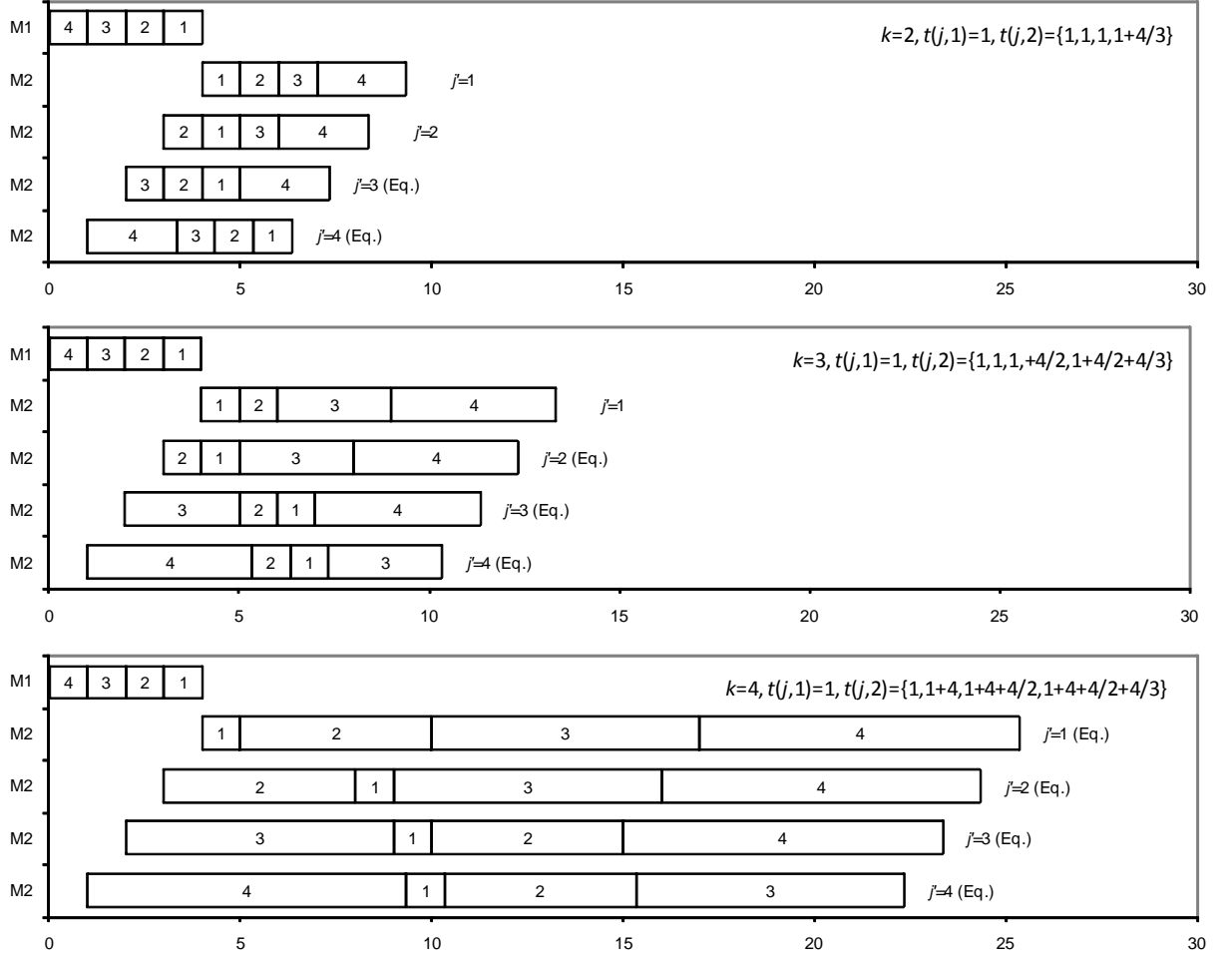
$$\bar{q}^{j'}(o, 2) = \begin{cases} j' - o + 1 & , \quad 1 \leq o \leq j' \\ o & , \quad j' + 1 \leq o \leq J, \end{cases}$$

and for  $J - k + 2 \leq j' \leq J$  defined by

$$\bar{q}^{j'}(o, 2) = \begin{cases} j' & , \quad o = 1 \\ J - k - o + 3 & , \quad 2 \leq o \leq J - k + 2 \\ o - 1 & , \quad J - k + 3 \leq o \leq j' \text{ and } k \geq 3 \\ o & , \quad j' + 1 \leq o \leq J, \end{cases}$$

where the lower two row cases are relevant only for some values of  $k$  and  $j'$ . The Gantt charts in

Figure 6 illustrate these sequences for  $J = 4$  and  $2 \leq k \leq 4$ .



$$\text{Lower bound} = \max \left\{ \frac{F_{k=2}^{\bar{q}^3} = F_{k=2}^{\bar{q}^4}}{F_{k=2}^{q^*}}, \frac{F_{k=3}^{\bar{q}^2} = F_{k=3}^{\bar{q}^3} = F_{k=3}^{\bar{q}^4}}{F_{k=3}^{q^*}}, \frac{F_{k=4}^{\bar{q}^1} = F_{k=4}^{\bar{q}^2} = F_{k=4}^{\bar{q}^3} = F_{k=4}^{\bar{q}^4}}{F_{k=4}^{q^*}} \right\}$$

$$= \max \left\{ \frac{58}{46}, \frac{88}{64}, \frac{172}{136} \right\} = \frac{11}{8} = 1.375$$

Figure 6: Gantt charts illustrating lower bound on maximal DC

Consider first  $\bar{q}^{j'}$  for  $1 \leq j' \leq J-k+1$ . Note that  $s_{\bar{q}^{j'}}(j', 2) = J-j'+1$ . For the job  $j'-o+1$  placed in the  $o^{\text{th}}$  position for  $o = 2$ , the starting time is  $s_{\bar{q}^{j'}}(\bar{q}^{j'}(o, 2), 2) = \max\{s_{\bar{q}^{j'}}(j'-o+1, 1) + t(j'-o+1, 1), s_{\bar{q}^{j'}}(\bar{q}^{j'}(o-1, 2), 2) + t(\bar{q}^{j'}(o-1, 2), 2)\} = \max\{(J-(j'-o+1))+1, (J-j'+o-1)+1\} = J-j'+o$ , i.e., with no idle time. By induction, this holds also for all jobs placed in  $3 \leq o \leq j'$ . In particular, job 1 in the  $(j')^{\text{th}}$  position starts at  $s_{\bar{q}^{j'}}(1, 2) = J$ , therefore all later jobs have already arrived to machine 2, thus can be processed with no idle times. Therefore, by the argument above,  $\bar{q}^{j'}$  is

optimal for machine 2 among all sequences starting with job  $j'$ . We can now compare the  $\bar{q}^{j'}$  for  $1 \leq j' \leq J - k + 1$ . Since all of these sequences follow SPT on machine 2 for all jobs, including job  $j'$ , minimizing the total flowtime  $F_2^{\bar{q}^{j'}}$  over  $1 \leq j' \leq J - k + 1$  is equivalent to minimizing the starting time,  $s_{\bar{q}^{j'}}(j, 2) = J - j' + 1$ . Thus the minimum is attained at the maximum value of  $j'$ , i.e.,  $J - k + 1$ . For this  $j'$ , since the starting time on machine 2 is  $s_{\bar{q}^{j'}}(j', 2) = J - j' + 1$  and all other jobs are scheduled with no idle times, the system flowtime  $F_1^{\bar{q}^{j'}} + F_2^{\bar{q}^{j'}}$  sums for all jobs the starting time  $J - j' + 1$  plus the remaining flowtime on machine 2 (as demonstrated in Figure 6 where  $j' = 3$  for  $k = 2$ ,  $j' = 2$  for  $k = 3$  and  $j' = 1$  for  $k = 4$ ). Thus  $F_1^{\bar{q}^{j'}} + F_2^{\bar{q}^{j'}}$  equals

$$\begin{aligned}
& (J - j' + 1)J + \sum_{o=1}^J (J - o + 1)t(\bar{q}^{j'}(o, 2), 2) \\
= & (J - j' + 1)J + 1 \cdot \sum_{j=1}^{J-k+1} (j + k - 1) + \sum_{j=J-k+2}^J (J - j + 1) \left(1 + \sum_{i=J-k+2}^j \frac{J}{i-1}\right) \\
= & kJ + \frac{1}{2}(J - k + 1)(J + k) + (k - 1) + \sum_{i=J-k+2}^J (J - i + 1) \frac{J}{i-1} \\
& + \sum_{j=J-k+2}^J (J - j) \left(1 + \sum_{i=J-k+2}^j \frac{J}{i-1}\right) \\
= & \frac{1}{2}(J - k + 1)(J + k) + (J + k - 1) + \sum_{i=J-k+2}^J \frac{J^2}{i-1} + \sum_{j=J-k+2}^J (J - j) \left(1 + \sum_{i=J-k+2}^j \frac{J}{i-1}\right) \\
= & \frac{1}{2}(J - k + 1)(J + k) + (J + k - 1) + \sum_{i=J-k+2}^J \frac{J}{i-1} (J + \sum_{j=i}^J (J - j)) + \sum_{j=J-k+2}^J (J - j) \\
= & \frac{1}{2}(J - k + 1)(J + k) + (J + k - 1) + \sum_{i=J-k+2}^J \frac{J}{i-1} \left(J + \frac{1}{2}(J - i + 1)(J - i)\right) \\
& + \frac{1}{2}(k - 1)(k - 2) \\
= & \frac{1}{4}J(4 + 2J + 3k - k^2) + \sum_{i=J-k+2}^J \frac{1}{i-1} J^2 \left(1 + \frac{1}{2}(J - i)\right).
\end{aligned}$$

Now consider  $\bar{q}^{j'}$  for  $J - k + 2 \leq j' \leq J$ . We show that there are still no idle time for  $\bar{q}^{j'}(2, 2) = J - k + 1$ . To see this note that  $s_{\bar{q}^{j'}}(J - k + 1, 2) = \max\{s_{\bar{q}^{j'}}(J - k + 1, 1) + t(J - k + 1, 1), s_{\bar{q}^{j'}}(j', 2) + t(j', 2)\} = \max\{(J - (J - k + 1)) + 1, (J - j' + 1) + (1 + \sum_{i=J-k+2}^{j'} \frac{J}{i-1})\} = (J - j' + 1) + (1 + \sum_{i=J-k+2}^{j'} \frac{J}{i-1})$ , where the last equality follows because  $\frac{J}{i-1} \geq 1$  for all  $J - k + 2 \leq i \leq j'$ , thus  $J - j' + 1 + (1 + \sum_{i=J-k+2}^{j'} \frac{J}{i-1}) \geq J - j' + 1 + (1 + j' - (J - k + 2) + 1) = k + 1 > k$ . For the job  $J - k - o + 3$  placed in the  $o^{\text{th}}$  position for  $o = 3$ ,  $s_{\bar{q}^{j'}}(\bar{q}^{j'}(o, 2), 2) = \max\{(J - (J - k - o + 3)) + 1, s_{\bar{q}^{j'}}(\bar{q}^{j'}(o - 1, 2), 2) + t(\bar{q}^{j'}(o - 1, 2), 2)\} = \max\{k + o - 2, s_{\bar{q}^{j'}}(\bar{q}^{j'}(o - 1, 2), 2) + 1\} = s_{\bar{q}^{j'}}(\bar{q}^{j'}(o - 1, 2), 2) + 1$ , where the last equality follows because we have shown that  $s_{\bar{q}^{j'}}(\bar{q}^{j'}(o - 1, 2), 2) \geq k + o - 3$ . Thus there is no idle time and  $s_{\bar{q}^{j'}}(\bar{q}^{j'}(o, 2), 2) = s_{\bar{q}^{j'}}(J - k + 1, 2) + o - 2$ . By induction, this holds

also for all jobs placed in  $4 \leq o \leq J - k + 2$ . In particular, job 1 in the  $(J - k + 2)^{\text{th}}$  position starts at  $s_{\bar{q}^{j'}}(J - k + 1, 2) + J - k > k + J - k = J$ , therefore all later jobs have already arrived to machine 2, thus can be processed with no idle times. Thus again,  $\bar{q}^{j'}$  is optimal for machine 2 among all sequences starting with job  $j'$ . As before, since  $s_{\bar{q}^{j'}}(j', 2) = J - j' + 1$  and all other jobs are scheduled with no idle times, the system flowtime  $F_1^{\bar{q}^{j'}} + F_2^{\bar{q}^{j'}}$  equals

$$\begin{aligned}
& (J - j' + 1)J + \sum_{o=1}^J (J - o + 1)t(\bar{q}^{j'}(o, 2), 2) \\
= & (J - j' + 1)J + (1 + \sum_{i=J-k+2}^{j'} \frac{J}{i-1})J + 1 \cdot \sum_{j=1}^{J-k+1} (j + k - 2) \\
& + \sum_{j=J-k+2}^{j'-1} (J - j)(1 + \sum_{i=J-k+2}^j \frac{J}{i-1}) + \sum_{j=j'+1}^J (J - j + 1)(1 + \sum_{i=J-k+2}^j \frac{J}{i-1}) \\
= & (J - j' + 1)J + (1 + \sum_{i=J-k+2}^{j'} \frac{J}{i-1})J + \frac{1}{2}(J - k + 1)(J + k - 2) \\
& + \sum_{j=J-k+2}^J (J - j)(1 + \sum_{i=J-k+2}^j \frac{J}{i-1}) + \sum_{i=j'+1}^J (J - i + 1) \frac{J}{i-1} \\
= & \frac{1}{2}(J - k + 1)(J + k) + (J + k - 1) + \sum_{i=J-k+2}^J \frac{J^2}{i-1} + \sum_{j=J-k+2}^J (J - j)(1 + \sum_{i=J-k+2}^j \frac{J}{i-1}),
\end{aligned}$$

which is independent of  $j'$  and equals one of the expressions we got for  $j' = J - k + 1$ . Therefore all  $\bar{q}^{j'}$  for  $J - k + 1 \leq j' \leq J$  give identical system flowtime, thus all are Nash equilibria with identical system flowtime (as demonstrated in Figure 6 where  $j' = 3, 4$  for  $k = 2$ ,  $j' = 2, 3, 4$  for  $k = 3$  and  $j' = 1, 2, 3, 4$  for  $k = 4$ ).

Finally, to compute the limiting DC, we need to establish an expression for the optimal system flowtime for  $\varepsilon \rightarrow 0^+$ . First we show that the optimal sequencing for the system,  $q^*$ , has all jobs sequenced on both machines according to SPT based on the process durations on machine 2. To see this note that for any sequence  $q$ , the first job on machine 2 starts no sooner than  $s_q(q(1, 2), 2) = 1$ . Since the order of the jobs on both machines according to  $q^*$  is identical and the processing durations on machine 1 are dominated by those on machine 2, i.e.,  $\min_j t(j, 2) \geq \max_j t(j, 1)$ , the jobs on machine 2 start exactly at  $s_{q^*}(1, 2) = 1$  with no idle times among the jobs. Therefore the system flowtime is  $J + \sum_{j=1}^J (J - j + 1)t(j, 2)$ , where the first component is the minimal possible starting time on machine 2 and the second component is the minimum flowtime of  $J$  jobs on machine 2.

Hence, this flowtime is optimal for the system and can be simplified to

$$\begin{aligned}
F_1^{q^*} + F_2^{q^*} &= J + \sum_{j=1}^J (J - j + 1)t(j, 2) \\
&= J + 1 \cdot \sum_{j=1}^{J-k+1} (J - j + 1) \\
&\quad + \sum_{j=J-k+2}^J (J - j + 1) \left(1 + \sum_{i=J-k+2}^j \frac{J}{i-1}\right) \\
&= J + \sum_{j=1}^J (J - j + 1) + \sum_{i=J-k+2}^J \frac{J}{i-1} \sum_{j=i}^J (J - j + 1) \\
&= \frac{1}{2}J(J+3) + \sum_{i=J-k+2}^J \frac{1}{i-1} \frac{1}{2}J(J-i+1)(J-i+2).
\end{aligned}$$

This leads to the proposed expression for the lower bound on the maximal DC (as demonstrated at the bottom of Figure 6). ■

Looking at the structure of the solutions shown in Figure 6 one can gain some insights regarding the typical instances that lead to a relatively high DC value in a flowshop system. This structure has the following two properties: (1) there is an inverse relation between the processing durations of the two machines; and (2) the processing durations of machine 1 are almost identical while there are significant differences between the processing durations of machine 2. As a result, following the SPT order on machine 1, in equilibrium machine 2 has to either process the jobs in a SPT sequence while starting processing relatively late, or start processing the jobs earlier but not in a SPT order. To the contrary, in the optimal solution machine 1 may need to process the jobs in a LPT order so that machine 2 will be able to start processing earlier in a SPT order. In this case the increase in flowtime for machine 1 will be very small due to its almost identical processing durations, while the improvement for machine 2 will be significant, resulting in improvement for the system.

Although the lower bound in Theorem 3 approaches 1 when  $J$  goes to infinity, the value of the lower bound is quite high even for large values of the number of jobs,  $J$ . For example, numeric calculation shows that it is equal to 1.08746 for  $J = 5000$ . Figure 7 illustrates the lower bound as a function of  $J$  for  $2 \leq J \leq 100$ , and of the ratio within the lower bound as a function of  $k$  for  $J = 5000$  and  $2 \leq k \leq J$ .

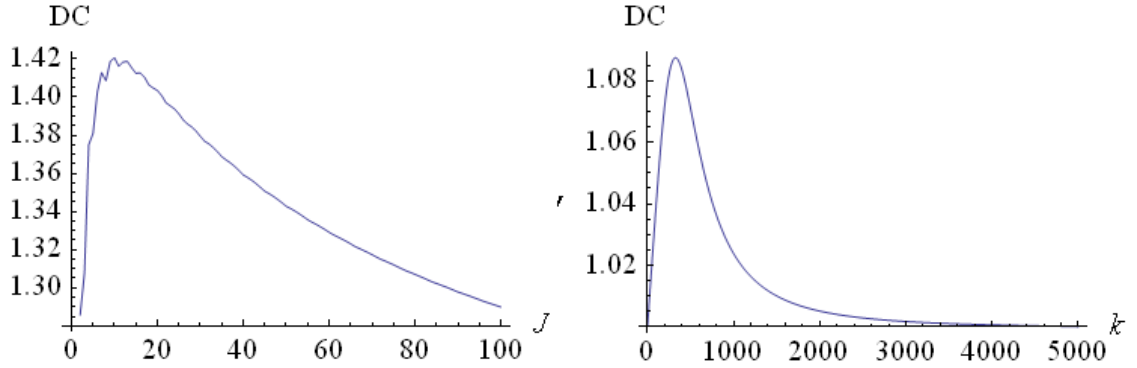


Figure 7: A lower bound for the maximal DC as a function of (a) the number of jobs  $J$ , and (b) the parameter  $k$  for  $J = 5000$  jobs.

## 5 A simple mechanism for coordination

In this section we investigate a simple mechanism for improving the system efficiency and reducing the DC. In discussing possible mechanisms, one should note that cost sharing based on the system flowtime may be potentially used to eliminate the inefficiency. However, such a mechanism requires a binding agreement that involves monetary transfers between the DMs after any sequencing decision they might make. Such an agreement may not be easy to implement in some environments. Therefore we propose instead a mechanism based on penalizing a machine found to have a simple possible change within its chosen sequence, leading to improvement for the system. By a simple change we mean a swap of only two jobs in the sequence chosen by the machine. We call it a simple change because finding whether such a change exists is a computation task of polynomial complexity as a function of the number of jobs in the system. Whenever such a swap is found, the machine is penalized. Such penalties can be achieved by postponing the departure of the jobs from the machine, resulting in a very high flowtime for this machine. Alternatively, monetary penalties may be imposed on the manager responsible for the machine. Note that this penalty method has the advantage that it is easy to implement, and moreover only takes the role of a threat which is never actually used.

We investigate the proposed mechanism theoretically in this section, and provide further empirical analysis in Section 6. Both analyses show the advantages of the mechanism, despite its heuristic nature. In particular, the empirical analysis of Section 6 demonstrates that the mechanism performs well in reducing the DC. Moreover, although one may expect that the mechanism

can increase the DC for some instances, this was not observed in the empirical study, suggesting that it is not likely to happen.

The following theorem shows that the mechanism completely solves the inefficiency existing in the settings investigated in Sections 3 and 4. Note that although these settings lead to a significantly high DC values, the mechanism reduces the DC to 1 for any number of jobs possible in the system.

**Theorem 4** *Under the proposed simple mechanism, the DC equals 1 for the jobshop and flowshop lower bound settings of theorems 2 and 3.*

**Proof.** For each of the settings, it is sufficient to show that there exists an optimal sequence for the system, such that for any improving deviation away from this sequence by a single machine while the other machine still follows the sequence, there exists a single swap of two jobs on the deviating machine that leads to improvement for the system. Showing this implies that any machine prefers not to deviate from the optimal sequence for the system because either this deviation is not improving or it leads to a very high penalty. As a result, the optimal sequence for the system is a Nash equilibrium, thus the DC equals 1.

We first show the property above for the jobshop setting of Theorem 2. Fix any optimal type  $O$  sequence. In the corresponding schedule there is a sequence of blocks of jobs, each consisting of two jobs that start their processing simultaneously on the two machines, then switch machines and complete their processing also simultaneously. Suppose that a machine  $\bar{m}$  deviates alone from the type  $O$  sequence and let block  $i$  be the first block of jobs for which this deviation occurs. There are four cases to consider, depending on the position (either first or second) within block  $i$  of the first deviating job,  $j_2$ , which replaces a job  $j_1$  in the type  $O$  sequence, and on the identity of the last machine on job  $j_2$ 's processing order. For each case we show that swapping  $j_1$  and  $j_2$  decreases the system flowtime by reducing the sum of system completion times of the jobs sequenced from block  $i$  onwards (while keeping unchanged the system completion time of all preceding jobs).

Case 1: The deviating job  $j_2$  is first in block  $i$  on machine  $\bar{m}$  and belongs to this machine. Since  $j_1$  is sequenced second in block  $i$  on the other machine, swapping  $j_1$  and  $j_2$  on  $\bar{m}$  decreases the system completion time of any job that was delayed before the swap due to the idle time forced on the other machine while it waits for  $j_1$  to arrive. Moreover, the system completion time of no job is increased by the swap because the only job that can potentially be delayed,  $j_2$ , is sequenced after  $j_1$  on the other machine, which determines the system completing time of both  $j_1$  and  $j_2$ , and so the system completion time of  $j_2$  can only be decreased by the swap. Therefore the system

flowtime decreases.

Case 2: The deviating job  $j_2$  is first in block  $i$  on machine  $\bar{m}$  and belongs to the other machine. To be a feasible deviation for machine  $\bar{m}$  (i.e., each machine does not wait without bound for the arrival of a job from the other machine), job  $j_2$  must equal the job sequenced first in block  $i$  on the other machine, thus leaving  $\bar{m}$  idle while the other machine is processing  $j_2$ . For the same reason of feasible deviation for  $\bar{m}$ , since the other machine has job  $j_1$  sequenced second in block  $i$ , job  $j_2$  must be followed on  $\bar{m}$  by a sequence of jobs ending with  $j_1$  such that each job in the sequence belongs to  $\bar{m}$ . As in Case 1, swapping  $j_1$  and  $j_2$  on machine  $\bar{m}$  decreases the system completion time of any job that was delayed before the swap due to the idle time forced on the other machine while it waits for  $j_1$  to arrive. The only difference from Case 1 is that the swap now may increase the system completion time of  $j_2$  because  $\bar{m}$  is last in this job's processing order. However, after the swap, job  $j_1$  can fill the free time slot that appeared before the swap on  $\bar{m}$  at the beginning of block  $i$  before  $j_2$ , while  $j_2$  just takes the time slot that was previously occupied by  $j_1$ . Thus the system completion time of  $j_1$  decreases by more than the corresponding increase for  $j_2$ . Therefore the system flowtime decreases.

Case 3: The deviating job  $j_2$  is second in block  $i$  on machine  $\bar{m}$  and belongs to this machine. We consider two subcases. Suppose first that machine  $\bar{m}$  is idle during some time interval before it starts processing job  $j_1$ . Job  $j_3$  immediately following this idle interval must belong to the other machine, otherwise it could have been processed earlier. Swapping  $j_1$  and  $j_3$ , the system completion time of  $j_1$  decreases by more than the increase in the system completion time of  $j_3$ . Since both  $j_1$  and  $j_3$  belong to the other machine, there is no increase in the system completion time of any other job, therefore the system flowtime decreases. Now suppose that  $\bar{m}$  is not idle before processing  $j_1$ . Due to the type  $O$  sequence on the other machine, the number of jobs belonging to the other machine that complete processing before  $j_1$  is at most  $\lfloor \frac{r}{2} \rfloor - 1$ , where  $r$  is the starting time of  $j_1$ . Thus the number of  $\bar{m}$  jobs that start processing before  $j_1$  is  $\lfloor \frac{r}{2} \rfloor + 1$ . Since there is no idle interval on machine  $\bar{m}$  before  $j_1$ , one of the  $\bar{m}$  jobs that start processing before  $j_1$ , say  $j_3$ , must complete processing after  $j_1$ . Swapping  $j_1$  and  $j_3$ , the system completion time of  $j_1$  decreases with no increase in the system completion time of any other job, including  $j_3$ , therefore the system flowtime decreases.

Case 4: The deviating job  $j_2$  is second in block  $i$  on machine  $\bar{m}$  and belongs to the other machine. Since job  $j_2$  is processed after block  $i$  on the other machine, there is an idle interval on  $\bar{m}$  while it waits for  $j_2$  to arrive. Swapping jobs  $j_1$  and  $j_2$  on  $\bar{m}$ , since both jobs belong to the other machine,

the swap does not delay the system completion time of any job. Moreover, the system completion time of  $j_1$  is decreased because it is now scheduled at the beginning of the idle interval that occurred for  $\bar{m}$  before the swap. Therefore the system flowtime decreases.

We now prove the result for the flowshop setting of Theorem 3. We show the property described at the beginning of this proof also for this setting. Consider an optimal sequence, identical for both machines, beginning with SPT according to machine 1 where machine 2 is indifferent, and continuing with SPT according to machine 2 (where durations are decreasing for machine 1). First note that machine 2 will never deviate from this optimal sequence because for any sequence of machine 1, let alone the optimal sequence, machine 2 minimizes the system flowtime minus the sum of completion times on machine 1, where the latter is constant with respect to machine 2's strategy. Therefore it is sufficient to prove that the mechanism allows no profitable deviations for machine 1. Any deviation of machine 1 that improves for this machine but worsens for the system must delay processing on machine 2 because the sequence of jobs on machine 2 has not changed. Consider the first job of machine 2 that is delayed compared to the optimal schedule described above. Machine 2 must therefore be idle before processing this job. Moreover, this job cannot be first on machine 1, because that would not cause a further delay on machine 2. Consider swapping on machine 1 this job with the one immediately preceding it. Such a swap decreases the flowtime for machine 2 by at least 1 time unit due to the idle interval that occurred before the swap. Furthermore, the swap increases the flowtime for machine 1 by no more than  $\varepsilon \cdot J^2$  because no more than  $J$  jobs are delayed by no more than  $\varepsilon \cdot J$  time units. Since  $\varepsilon \cdot J^2 < 1$  as  $\varepsilon \rightarrow 0^+$ , the swap decreases the system flowtime. ■

## 6 Numerical experimentation

In the previous sections we showed that the maximal DC may be significantly large. Here, the DC is analyzed empirically via a wide set of experiments. The purpose of the experimentation is twofold: first, we study some characteristics of the DC, such as its average value and how it is affected by the problem parameters, and second, we examine the performance of the proposed mechanism on various instances. For each instance, the centralized optimal solution(s) as well as all pure Nash equilibria were calculated via exhaustive search. Due to scalability limitations, only small sized instances were studied (for example, in a 6-job instance, the number of cells in the game matrix is  $(6!)^2$ , so the number of comparisons needed for computing all Nash equilibria is

$(6!)^3 \simeq 3.7 \times 10^8$ ). Still, as discussed in the introduction, such settings may be found in projects management environments within real life organizations.

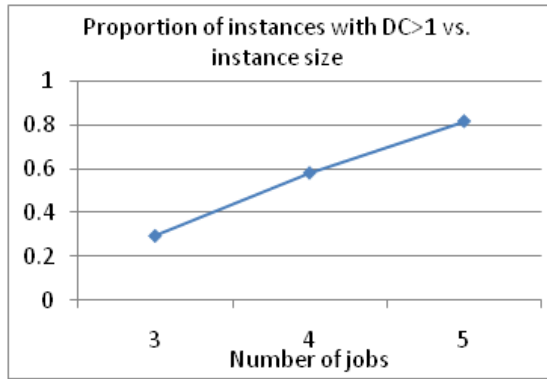
The analysis includes instances with three, four and five jobs. We distinguish between various types of instances according to the symbol  $(x-y)$ , where  $x, y$  denote the number of jobs belonging to machine 1, 2, respectively (i.e., jobs starting processing on the respective machine, then proceeding to the other machine). The following five types of instances were examined:  $(2-1)$ ,  $(3-1)$ ,  $(2-2)$ ,  $(4-1)$  and  $(3-2)$ . For each type, 500 instances were generated with random processing durations taken from a uniform distribution between 0 and 1. The results were analyzed according to two factors: the size of the instance in terms of the total number of jobs, and the degree of symmetry in the instance in terms of the number of jobs each machine owns. In asymmetric instance types one machine owns significantly more jobs than the other, hence is expected to have more power in the game, while in symmetric instance types both machines own about the same number of jobs, hence we expect both to have similar amount of power. We accordingly consider 4- and 5-job instances of types  $(3-1)$  and  $(4-1)$  as asymmetric, and types  $(2-2)$  and  $(3-2)$  as symmetric.

We found that around 60% of the instances are inefficient in the sense that the best decentralized solution (equilibrium) gives larger flowtime than that of the optimal solution ( $DC > 1$ ). This percentage value increases with the instance size (see Figure 8(a)), as 82% of the 5-job instances were inefficient. Consequently, we may conjecture that as the instance size increases, the optimal centralized solution will not likely be an equilibrium. The proportion of inefficient instances also increases as the problem becomes more symmetric, as demonstrated in Figure 8(b). This may be explained by the fact that when one machine has much more power than the other, it will have higher interest in the system global objective, leading to a solution minimizing simultaneously their own cost and the system cost.

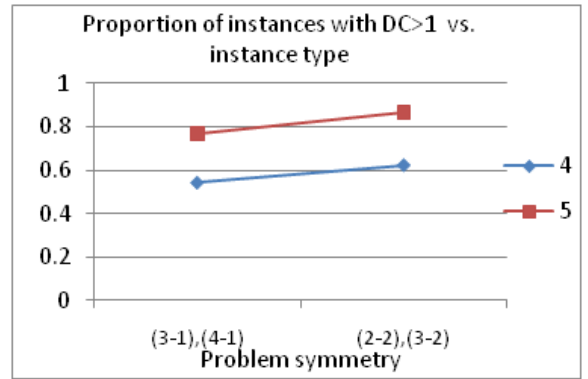
Among the inefficient problems, the average DC is 1.043, with a maximal value of 1.24. The average and maximal DC values increase as instances become more symmetric, as demonstrated in Figure 8(c) and 8(d). Along with the previous result, we may conclude that symmetric instances, in which players are equally strong, lead to a higher likelihood of inefficiency and a higher DC.

Additional results indicate that a unique optimal solution was obtained in 79% of the instances, and this value decreases with the instance size, as demonstrated in Figure 8(e). This result is quite expected since the feasible set increases with the instance size, thus there is higher likelihood of finding multiple optima. When analyzing the existence of pure equilibria, the results show that 82% of the instances have a single unique equilibrium, 7% have multiple equilibria and 11% have

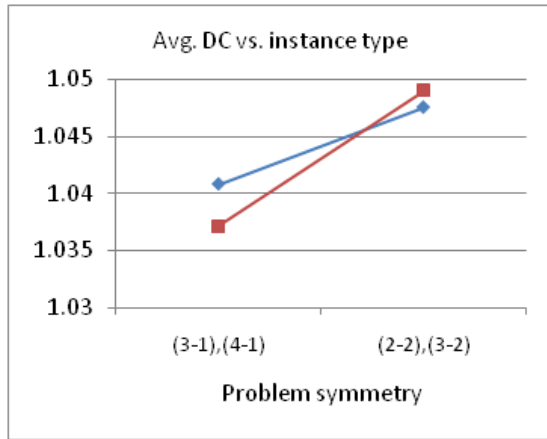
no equilibrium. The percentage of instances with a unique equilibrium decreases with the instance size, as demonstrated in Figure 8(f). Interestingly, most of this decrease is compensated by an increase in the percentage of instances with no pure equilibrium.



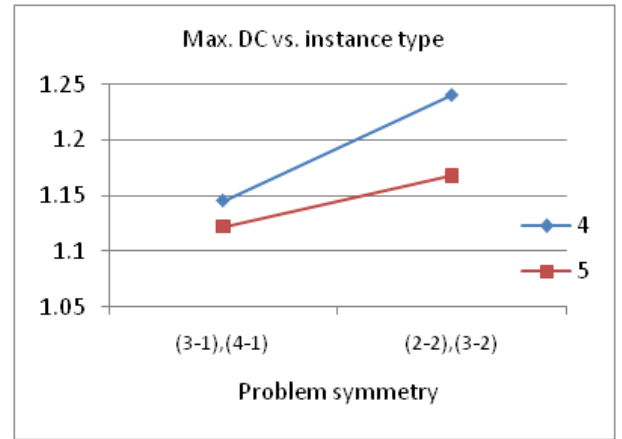
(a)



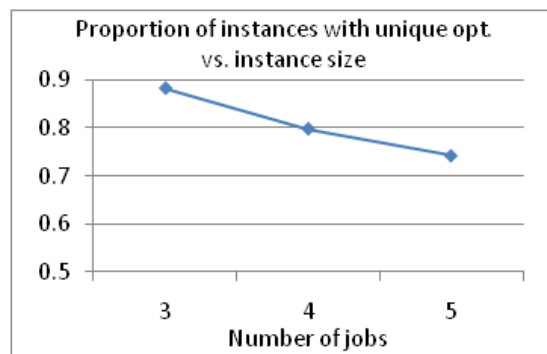
(b)



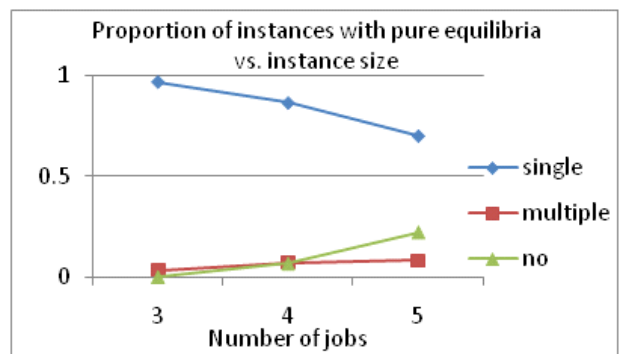
(c)



(d)



(e)



(f)

Figure 8: Graphical demonstration of experiment results

Another interesting result relates to the equilibrium characteristics. In around 91% of the problems, exactly one of the machines improves its utility versus the optimal solution. However, in the remaining instances, a lose-lose result is obtained in the sense that both machines worsen their utility compared to the optimal solution. This percentage increases with the instance size, as demonstrated in Figure 9(a). Another observation, as demonstrated in Figure 9(b), indicates that when some machine owns significantly more jobs than the other, this machine improves its utility equilibrium significantly more often than the other. This result supports the reasoning above regarding the relative advantage of having more power by owning more jobs (at the expense of the other machine).

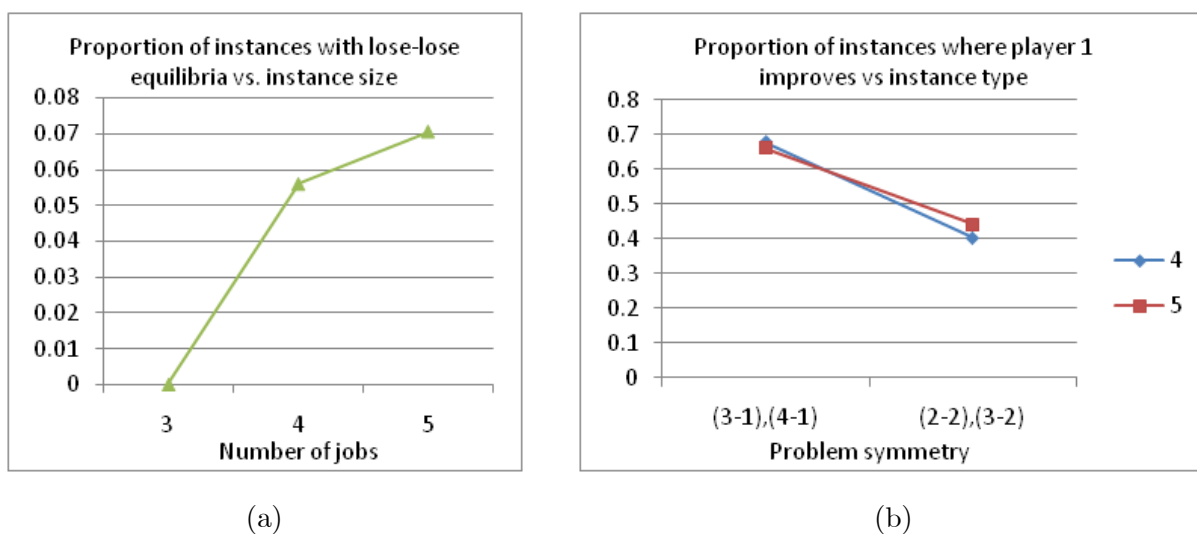


Figure 9: Additional demonstration of experiment results

The last set of results relates to the mechanism proposed in Section 5. The mechanism was applied to 200 instances of each type. As shown in Figure 10, the percentage of inefficient instances increases up to 78% without the mechanism, and up to only 9% when the mechanism is applied. The efficiency and improvement achieved with the mechanism both decrease with the instance size, but are still significant. We may conclude that the state of the system can be significantly improved

when using a simple polynomial mechanism.

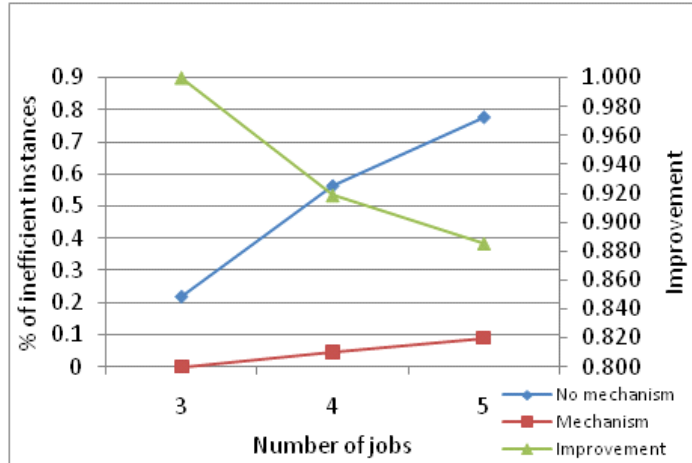


Figure 10: Percentage of inefficient instances

## 7 Concluding remarks

This paper investigates the multiple jobs, two machines jobshop problem modeled as a non-cooperative game, in which the sequencing decisions for each machine are taken independently by the respective machine owner. Such situations may be found in the operational level of manufacturing and service systems, where jobs or projects are processed by different machines or organization resources/functions, and each resource manager can determine the job processing sequences within their territory. Decentralized decisions are expected to provide inferior solutions compared to the centralized environment, mostly because the incentives of each decision maker are not necessarily aligned with the centralized objective. We examined the potential system loss due to the decentralized decision making process, which is expressed by the decentralization cost (DC). We analyzed a setting where single players as well as the system aim at minimizing the same flowtime objective and showed that the system loss can be still significant.

After presenting our model and demonstrating the optimal (centralized) and equilibrium (decentralized) schedules using small examples, we showed that system efficiency can be guaranteed only for the smallest instance of two jobs that start their processing on different machines and proceed to the other machine. Then, to examine the possibility of high DC values, we developed closed form expressions for lower bounds on the maximal DC, both for jobshop and flowshop settings. For jobshop, the lower bound was achieved using a symmetric system with an arbitrary even number

of jobs with equal processing time, half belonging to machine 1 and the other half belonging to machine 2. The DC obtained for this system, which is a lower bound on the maximal DC expected for general instances, approaches 1.5 asymptotically as the number of jobs increases to infinity, with relatively high values for small instances (value of 1.3 for 8 jobs). For flowshop settings, our lower bound on the maximal DC shows high values for small instances as well as for large scale systems (above 1.08 for a 5,000 job instance), with a maximum value of  $\frac{6809}{4793} \simeq 1.4206$  for 10 jobs. We then developed a simple mechanism based on pairwise exchange for improving the efficiency of the system. We showed that the mechanism is able to coordinate the job shop and the flowshop settings for which our lower bounds were developed. Empirical results indicate that the mechanism is highly efficient in general job shop problems.

A wide experimentation of relatively small problems provided some insights regarding the DC value. We found that most problems are inefficient and the DC value increases with the instance size. An average DC of 1.043 was obtained, with a maximal value of 1.24. When comparing symmetric versus asymmetric instances in terms of the number of jobs owned by each machine, we found that symmetry leads to higher likelihood of inefficiency and higher DC values. Therefore a symmetric system in which machines have roughly the same power will perform in lower efficiency than a system with a dominant player. The reason may be that under asymmetry, the interest of the dominant player will be more aligned with the system objective in the expense of the other player. When comparing the equilibrium solution to the optimal one, we found that in 91% of the cases exactly one of the machines improves its utility in equilibrium as compared to the optimal solution, where the improving machine is likely to be the one with the larger number of jobs in asymmetric instances. The remaining instances show a lose-lose situation, where both machines worsen their utility.

## References

- Anshelevich, E., A. Dasgupta, J. Kleinberg, É. Tardos, T. Wexler, T. Roughgarden. 2004. The Price of Stability for Network Design with Fair Cost Allocation, FOCS '04 Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science.
- Aydinliyim, T., G.L. Vairaktarakis. 2010a. Coordination of Outsourced Operations to Minimize Weighted Flow Time and Capacity Booking Costs. *Manufacturing & Service Operations Management*, 12 (2), 236-255.

- Aydinliyim, T., G.L. Vairaktarakis. 2010b, to appear. Sequencing Strategies and Coordination Issues in Outsourcing and Subcontracting Operations. K. Kempf, P. Keskinocak, R. Uzsoy, eds. Handbook of Production Planning. Kluwer Academic Publishers.
- Borm, P., H. Hamers, R. Hendrickx. 2001. Operations Research Games: A Survey. CentER Discussion Paper #2001-45, Tilburg University, Tilburg, The Netherlands.
- Borm, P., G. Fiestras-Janeiro, H. Hamers, E. Sánchez, M. Voorneveld. 2002. On the convexity of games corresponding to sequencing situations with due dates. *European Journal of Operational Research* 136, 616–634.
- Bukchin, Y., E. Hanany. 2007. Decentralization Cost in Scheduling: A Game-Theoretic Approach. *Manufacturing & Service Operations Management*, 9(3), 263-275.
- Christodoulou, G., E. Koutsoupias, A. Nanavati. 2004. Coordination mechanisms. *Automata, Languages and Programming*, Vol. 3142 of Lecture Notes in Computer Science, pp. 345–357. Springer, Berlin.
- Correa, J.R., M. Queyranne. 2010. Efficiency of Equilibria in Restricted Uniform Machine Scheduling with Minsum Social Cost. Mimeo, University of Chile.
- Curiel, I., H. Hamers, F. Klijn. 2002. Sequencing Games: a Survey. P. Borm, H. Peters, eds. *Chapters in Game Theory in Honor of Stef Tijs*. Kluwer, Boston, MA. 27-50.
- Curiel, I., G. Pederzoli, S. Tijs. 1989. Sequencing Games. *European Journal of Operational Research* 40 344-351.
- Hain, R., M. Mitra. 2004. Simple Sequencing Problems with Interdependent Costs. *Games and Economic Behavior* 48 271-291.
- Hall, N.G., Z. Liu. 2008. Cooperative and Noncooperative Games for Capacity Planning and Scheduling. *Tutorials in Operations Research INFORMS* 2008, 108-129.
- Hamers, H., P. Borm, S. Tijs. 1995. On Games Corresponding to Sequencing Situations with Ready Times. *Mathematical Programming* 70, 1–13.
- Hamers, H., F. Klijn, J. Suijs. 1999. On the Balancedness of Multimachine Sequencing Games. *European Journal of Operational Research* 119(3) 678–691.

- Harland, C.M., R.C. Lammings, J. Zheng, T.E. Johnsen. 2001. A Taxonomy of Supply Networks. *Journal of Supply Chain Management* 37(4), 21-27.
- Heydenreich, B., R. Müller, M. Uetz. 2007. Games and Mechanism Design in Machine Scheduling – An Introduction. *Production and Operations Management* 16(4), 437–454.
- Immorlica, N., L. Li, V.S. Mirrokni, A. Schulz. 2005. Coordination mechanisms for selfish scheduling. X. Deng and Y. Ye, eds. *Internet and Network Economics*, Vol. 3828 of *Lecture Notes in Computer Science*, pp. 55–69. Springer, Berlin, Germany.
- Jéhiel, P., B. Moldovanu. 2001. Efficient Design with Interdependent Valuations. *Econometrica* 69 1237-1259.
- Koutsoupias, E., C. Papadimitriou. 1999. Worst-case equilibria. C. Meinel and S. Tison, eds. *STACS 1999, 16th Annual Symposium on Theoretical Aspects of Computer Science*, Vol. 1563 of *Lecture Notes in Computer Science*, pp. 404–413. Springer, Berlin, Germany.
- Kutanoglu, E., S.D.Ž. Wu, 1999. On Combinatorial Auction and Lagrangean Relaxation for Distributed Resource Scheduling, *IIE Transactions* 31, 813-826.
- Lenstra, J.K., A.H.G. Rinnooy Kan, P. Brucker. 1977. Complexity of machine scheduling problems, *Annals of Discrete Mathematics* 1, 343-362.
- Nisan, N., A. Ronen. 2001. Algorithmic mechanism design. *Games and Economic Behavior* 35, 166–196.
- Pinedo, M. 2002. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, Englewood Cliffs, NJ.
- Reeves, D. M., M. P. Wellman, J. K. MacKie-Mason, A. Osepayshvili. 2005. Exploring Bidding Strategies for Market-Based Scheduling. *Decision Support Systems*, 39, 67–85.
- Rosenthal, R.W. 1973. A class of games possessing pure-strategy Nash equilibria, *International Journal of Game Theory* 2, 65–67.
- Slikker, M. 2006. Relaxed Sequencing Games Have a Nonempty Core. *Naval Research Logistics* 53, 235-242.

Wellman, M.P., W. E. Walsh, P. R. Wurman, J. K. MacKie-Mason. 2001. Auction Protocols for Decentralized Scheduling. *Games and Economic Behavior* 35, 271-303.