International Journal of Production Research
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tprs20

Batch scheduling on parallel machines with dynamic job arrivals and incompatible job families

Eray Cakici [a], Scott J. Mason [b], John W. Fowler [c] & H. Neil Geismar [d]

[a] IBM, Yapi Kredi Plaza, Levent, Istanbul, Turkey

[b] Department of Industrial Engineering, Clemson University, Clemson, USA

[c] Department of Industrial Engineering, Arizona State University, Tempe, USA

[d] Department of Information and Operations Management, Mays Business School, Texas A&M University, College Station, USA

Version of record first published: 03 Jan 2013.

To cite this article: Eray Cakici , Scott J. Mason , John W. Fowler & H. Neil Geismar (2013): Batch scheduling on parallel machines with dynamic job arrivals and incompatible job families, International Journal of Production Research, DOI:10.1080/00207543.2012.748227

To link to this article: http://dx.doi.org/10.1080/00207543.2012.748227

International Journal of Production Research
2012, 1–16, iFirst Article

Batch scheduling on parallel machines with dynamic job arrivals and incompatible job families

Eray Cakici[a]*, Scott J. Mason[b], John W. Fowler[c] and H. Neil Geismar[d]

[a]*IBM, Yapi Kredi Plaza, Levent, Istanbul, Turkey;* [b]*Department of Industrial Engineering, Clemson University, Clemson, USA;* [c]*Department of Industrial Engineering, Arizona State University, Tempe, USA;* [d]*Department of Information and Operations Management, Mays Business School, Texas A&M University, College Station, USA*

(*Received 7 November 2011; final version received 14 November 2012*)

We study the scheduling problem of minimising weighted completion times on parallel identical batching machines with dynamic job arrivals and incompatible job families. Each job is associated with a family, weight (priority), release time, and size. Batching machines can process simultaneously up to a specified total size of the jobs of a particular family. The scheduling problem can be represented by $Pm|r_j, batch, fmls|\sum w_jC_j$. We present a mathematical model and heuristic algorithms employing different local search procedures individually and sequentially under a variable neighbourhood search scheme. We have shown that among local searches, repositioning the batches instead of jobs yields better results. The best-performing heuristic algorithm is capable of generating solutions within 0.6% of the best overall heuristic solution for each instance in a reasonable amount of time. When this heuristic is compared against the mathematical model, solutions that are 3.7% above optimal on average in the 15-job problem instances are possible.

Keywords: scheduling; heuristics; batch scheduling; mathematical modelling; variable neighbourhood search

1. Introduction

Batch scheduling problems arise in many industries, such as semiconductor manufacturing, aircraft industries, steel casting, transportation, material handling, packaging, and storage systems. A batch is a group of items (jobs) that are processed simultaneously using the same resource (e.g. machine), on which batch size limitations typically exist. Batch capacities are mainly defined either as the number of jobs or total sizes of jobs that can be processed simultaneously. Batch scheduling problems are a combination of assignment and sequencing problems. The two main tasks are (1) forming batches (assigning jobs to batches) and (2) establishing the processing sequences on the batching machine(s). It is critical to achieve a good trade-off between the objectives of high machine utilisation, short lead times, and on-time delivery. Determining optimal batch formation and sequence can yield substantial savings by improving the utilisation of limited resources and reducing the number of batches, especially in the existence of longer processing times. In many production environments, batches can only include jobs of the same family (incompatible job family cases) because jobs of a particular family require the same processing recipe and time. For example, due to the diffusion operation's chemical nature in the wafer fabrication stage of semiconductor manufacturing, only jobs of the same family may be processed together because they require the same procedure (Uzsoy 1995). In family scheduling models, a setup time is not required for every job in a batch. There is only one setup per batch, and completion time of a job is equal to the completion time of its batch. (Potts and Kovalyov 2000).

This paper addresses the problem of loading and scheduling batching machines in an environment with job release times and incompatible job families. Our study is motivated by the fact that batch scheduling models are applicable to the distribution stage of supply-chain scheduling problems. In this environment, transportation units can be viewed as batching machines in which a group of jobs (orders) are processed (transported) together. Jobs become available for pickup at different times and are delivered to customers by capacitated vehicles. Each job is associated with a customer, and every customer represents a different job family. Only jobs of the same family can be grouped together. In other words, only jobs destined for the same customer are delivered together. Because production schedules could supply release times for the jobs in a distribution scheduling problem in a supply-chain environment, we consider different job release times. Figures 1 and 2 illustrate an example problem and an example schedule representation of integrated

*Corresponding author. Email: eray.cakici@gmail.com

ISSN 0020-7543 print/ISSN 1366-588X online

http://dx.doi.org/10.1080/00207543.2012.748227
http://www.tandfonline.com

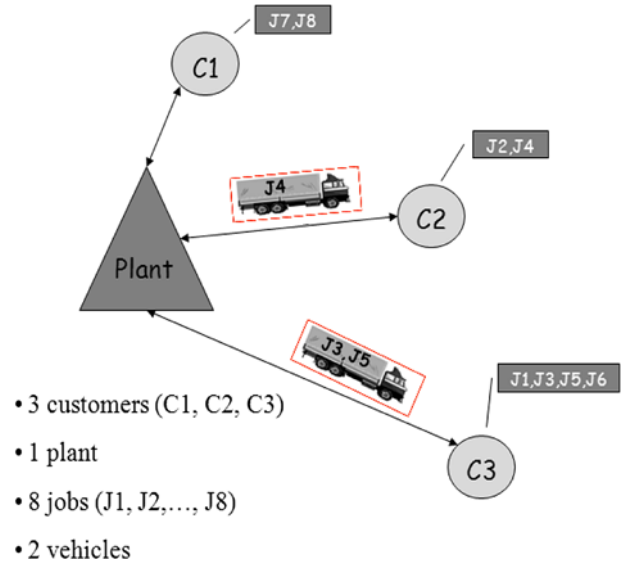

Figure 1. Example problem.

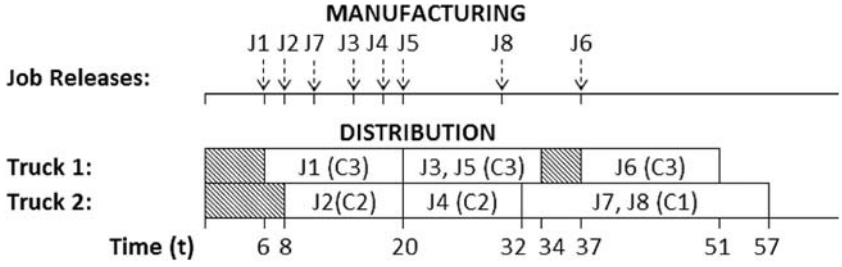

Figure 2. Example problem schedule.

manufacturing and distribution operations. As it is practically motivated, our problem also involves different job sizes (volumes or space required in the transportation unit) and job weights (priorities). In order to capture the service times and different job types, we study the performance measure of minimising total weighted completion time. As a result, the scheduling problem can be represented by $Pm|r_j, batch, fmls|\sum w_jC_j$ using the $\alpha|\beta|\gamma$ notation in Pinedo (2012). We provide a mathematical model that can solve small problem instances to optimality. We also develop efficient heuristic approaches and compare their performance to optimal solutions for small problem instances and to each other for larger instances.

A number of researchers have studied scheduling batching machines with incompatible job families. One body of the literature assumes that all jobs are available simultaneously – the *static* arrivals case. Azizoglu and Webster (2001) study minimising weighted completion time with the assumption that all jobs are ready at the beginning of the planning horizon and provide a branch-and-bound algorithm to obtain optimal solutions on a single batching machine. However, the algorithm cannot solve large (of more than 25 jobs) problem instances to optimality.

Perez, Fowler, and Carlyle (2005) present several heuristics to minimise weighted tardiness on a single batching machine. This study is extended in Erramilli and Mason (2008) by considering multiple orders per job. A number of heuristic algorithms are proposed. Orders are first grouped into jobs, and then jobs are grouped into batches by different rules in a single batch-processing machine system.

As we consider in this study, in many cases jobs will be available at different times of the planning horizon – the *dynamic* arrivals case. In such situations it can be advantageous to delay the processing of a batch in order to accommodate jobs arriving soon. Fowler, Phillips, and Hogg (1992) use the knowledge of future arrivals in order to develop an efficient heuristic algorithm to reduce the average time that lots (jobs) spend waiting in the queue at a batching machine. The trade-off between a total delay caused in the current batch by waiting for the next job and a possible delay eliminated in the subsequent jobs is evaluated in their approach. Uzsoy (1995) examines different scheduling problems and

proposes heuristics to minimise makespan, maximum lateness, and total weighted completion time on a single batching machine. Both the dynamic job arrivals case and the static case are investigated. Lee and Uzsoy (1999) study dynamic job arrivals on a single batching machine for minimising makespan. The authors present polynomial and pseudo-polynomial time algorithms for several special cases and efficient heuristics for the general problem. Sung et al. (2002) propose a dynamic programming algorithm for the same problem motivated by burn-in ovens in semiconductor manufacturing industry. Kurz and Mason (2008) develop an efficient heuristic algorithm for the problem of minimising total tardiness on a single batching machine considering job release times. Cheng and Kovalyov (2001) also propose efficient algorithms for special cases of single machine batch scheduling problem with different objective functions. Jobs are processed sequentially in a batch and processing time of a batch is the sum of processing times of jobs contained therein.

Another line of research focuses on parallel machine scheduling problems, as does our problem. Balasubramanian et al. (2004) consider the parallel batch machine scheduling problem to minimise total weighted tardiness. Two different versions of a genetic algorithm (GA) are proposed, and all jobs are associated with no release times. Mönch et al. (2005) study minimising total weighted tardiness on parallel machines with release times. Two different decomposition approaches are presented. Machine assignments are performed using GAs in which dispatching and scheduling rules are employed to build batch formations and sequences. Malve and Uzsoy (2007) develop an efficient GA for the parallel batch machine scheduling problem, minimising maximum lateness with dynamic job arrivals. In contrast to our study, in all of the above-mentioned studies researchers either assume all jobs have the same size and define batch capacity as the number of jobs that can be processed simultaneously or assume batches have an infinite capacity.

In particular, there are two papers in the literature that are closely relevant to our study. Dobson and Nambimadom (2001) investigate the problem of minimising total weighted completion time on a single batching machine with incompatible job families and jobs of different sizes. The problem is proven to be NP hard, and a number of heuristics are proposed along with a lower bound achieved through an integer programming formulation. Koh et al. (2005) extend this study by considering different objective functions, including makespan and total completion time. The authors present a number of dispatching algorithms and a genetic algorithm. Both papers consider different job sizes and define the machine capacity as the total job size that can be loaded simultaneously. Our problem extends the previous research by involving multiple machines and job release times while considering different job sizes as well.

The rest of the paper is organised as follows. A detailed problem description is provided in Section 2. In Section 3, we describe the mathematical programming formulation. Section 4 details heuristic solution algorithms based on employing different local search procedures individually and sequentially under a variable neighbourhood search scheme. In Section 5, we present the design of computational experiments, and results are discussed in Section 6. We conclude the paper with a summary and directions for future research.

2. Problem description

We emulate an actual supply-chain environment in which jobs arrive for delivery at various points in time during a production shift. First, the pertinent notations for the mathematical formulation are introduced:

J the set of jobs (orders) such that $J \in \{1, 2, \cdots, n\}$
M the set of machines (vehicles) $M \in \{1, 2, \cdots, m\}$
B the set of batches (vehicle trips) such that $B \in \{1, 2, \cdots, n\}$
T the set of time periods in the planning horizon $T \in \{1, 2, \cdots, \phi\}$.
Each job $j \in J$ has the following parameters associated with it:
w_j the weight of job $j \in J$
r_j release time of job $j \in J$
$\mathrm{v_j}$ volume (size) of job $j \in J$.

Each job is also ordered by a specific customer, and it has to be delivered directly to that customer. Jobs can be delivered by any of the available vehicles, regardless of a job's associated customer. Vehicles are capacitated, and capacity is dependent solely on volume of the jobs, not on shape. The distribution environment is modelled as a parallel batch machine system. Each job is allocated in a batch (vehicle trip), and each batch is processed (delivered) by any of the m parallel machines (vehicles) in $M = \{1,...,m\}$. We assume vehicles can only visit one customer in each trip. Therefore, only jobs of the same customer can be grouped together in a batch (incompatible job families), and a batch become available for processing only if all jobs in that batch have been released. Finally, we assume that (1) jobs cannot be split across multiple batches, and (2) the time to deliver a batch to a customer is independent of the size or number of jobs in the batch.

Although we assume that the trucking function has no influence over the production schedule, i.e. the release times are completely exogenous, two lemmas can be proposed by assuming agreeable release times, i.e. $r_j < r_{j+1}$ implies $w_j \geq w_{j+1}$:
Let B_c be the set of possible vehicle trips to customer c and R_c be the set of jobs (orders) to be delivered to customer c. Lemma 1 limits the number of potential truck departure times that need to be considered.

Lemma 1: There is an optimal schedule in which trip $b \in B_c$ departs either at time r_j, for some job $j \in R_c$, or when a truck returns.

Proof: Suppose that there is some trip $b \in B_c$ whose last job is $j \in R_c$ and that trip b departs at time t, where t > r_j, and that some transporter was at the depot for some positive duration ε before t. This trip could have departed earlier, i.e. at max $\{r_j, t - \varepsilon\}$, without increasing the objective.

Note that Lemma 1 implies that each time a job is released, if a truck is available, then a go/no-go decision is made for the current batch of that job's customer.

Lemma 2 states that when a truck departs, it carries all available (and appropriate) jobs that fit.

Lemma 2: Given agreeable weights, the first truck to depart for customer c after time r_j will include job j, provided $j \in R_c$, job j fits in that trip and $v_j \geq v_{j+1}$.

Proof: Suppose there is an optimal schedule S in which the current trip, say $b_1 \in B_c$, does not include job j but does include job $j + 1$, and job j is on the following trip to this customer, say $b_2 \in B_c$, which departs at time $t \geq r_{j+1}$. Consider a different schedule S' in which jobs j and $j + 1$ are reversed. Let W be the total weight of the jobs on trip b_1 with neither job j nor job $j + 1$. The difference in cost between the two schedules is

$$C(S) - C(S') \geq (W + w_{j+1})r_{j+1} + w_j t - \lfloor (W + w_j)r_j + w_{j+1}t \rfloor \quad (1)$$

$$= W(r_{j+1} - r_j) + w_j(t - r_j) - w_{j+1}(t - r_{j+1}) \quad (2)$$

$$> W(r_{j+1} - r_j) + w_j(t - r_j) - w_{j+1}(t - r_j) \geq 0, \quad (3)$$

so schedule S cannot have a lower cost than schedule S'. Hence, S' must be optimal. The condition $v_j \geq v_{j+1}$ prevents the inclusion of job j in trip b_2 from allowing b_2 to hold more jobs than if it had job $j + 1$. Our main goal in this paper is to address general case. However, lemmas are provided to provide insights for some special cases of the problem which can be a topic of future research.

3. Mathematical model

3.1 *Mixed-integer programming formulation*

Our problem is formulated as a time-indexed mixed integer program (MIP). MIP formulations with time-indexed variables have been shown to be very effective on machine scheduling problems, especially in those with release times and small processing times (Keha Khowala, and Fowler 2009). Unlu and Mason (2010) evaluate different mathematical formulations and show superiority of time-indexed MIP formulation for parallel machine scheduling problem with the objective of minimising total weighted completion time. In addition to notations introduced before, the following notations are used in the model:

Λ	set of job-to-batch pairs: $(j, k) \in \Lambda$ indicates job j and batch k are destined to the same family
x_{ikt}	1 if batch $k \in B$ starts its processing on machine $i \in M$ at time $t \in T$; otherwise, 0
y_{jk}	1 if job $j \in J$ is assigned to batch $k \in B$, which requires $(j, k) \in \Lambda$; otherwise, 0
C_j	the completion time of job $j \in J$
Z_k	time at which batch $k \in B$ finishes its required processing which is max $\{C_j : y_{jk} = 1\}$
δ	machine loading capacity.

In order to increase the efficiency of the model, we start with pre-defining that each job can only be assigned to one of its own family's batches. Therefore, no additional constraints are required to be introduced in the model. Every batch

is pre-designated for use by a specific job family and can only include jobs of that family – the number of jobs in each family determines the number of batch designations by family. By this pre-definition, number of variables is also reduced as compared with the case of introducing batch assignment variables for all job-to-batch combinations.

The mathematical model is as follows:

Objective function

$$TWC = \sum_{j:j\epsilon\ J} w_j C_j.$$

The objective function of interest is to minimise the sum of total weighted completion time (TWC) of all jobs.

Constraints

$$\sum_{k=1}^{n} y_{jk} = 1 \quad \forall j \in J{:}(j,k) \in \Lambda \tag{4}$$

$$\sum_{j=1}^{n} v_j y_{jk} \leq \delta \quad \forall k \in B{:}(j,k) \in \Lambda \tag{5}$$

$$\sum_{i=1}^{m} \sum_{t=0}^{\phi-\beta_k} x_{ikt} = 1 \quad \forall k \in B \tag{6}$$

$$\sum_{k=1}^{n} \sum_{h=\max\{0,t-\beta_k\}}^{t-1} x_{ikh} \leq 1 \quad \forall i \in M, \forall t \in T \tag{7}$$

$$Z_k \geq \sum_{i=1}^{m} \sum_{t=1}^{\phi-\beta_k} (t+\beta_k) x_{ikt} \quad \forall k \in B \tag{8}$$

$$Z_k \geq (r_j + \beta_k) y_{jk} \ \forall j \in J, \forall k \in B{:}(j,k) \in \Lambda \tag{9}$$

$$C_j \geq Z_k - \phi(1 - y_{jk}) \ \forall j \in J, \forall k \in B{:}\ (\mathrm{j,k}) \in \Lambda. \tag{10}$$

Equation (4) ensures that jobs are assigned to one of the available batches that are eligible to include the corresponding job family. Machine (vehicle) capacity is taken into consideration in Equation (5). Then, Equation (6) forces batches to be assigned to exactly one of the m available machines with a starting time exactly at one of the possible time periods. In Equation (6), β_k is the processing time of batch $k \in B$ and number of families is denoted by Γ. ϕ is the upper bound on the completion time of the latest job and can be defined as $\max\{r_j : \forall_j \in J\} + \sum_{f=1}^{\Gamma} \lceil \Lambda_f/\delta \rceil p_f$. Λ_f is the total volume of the jobs, and p_f is the batch-processing time associated with family f. By considering all batches and all time periods a batch requires on any machine, Equation (7) ensures that two batches cannot be processed at the same time on the same machine. Equation (8) computes batch k's completion time Z_k as its processing start time plus its processing time. Equation (9) ensures that a batch cannot start its processing until all jobs assigned to the corresponding batch become ready. Finally, in Equation (10), a job's completion time is determined by the completion time of the batch to which it is assigned.

3.2 *Problem complexity*

The scheduling problem studied here consists of parallel machines and a set of jobs with different release times and incompatible job families. Each job has to be processed in any of the m parallel batching machines. Jobs of the same family (designated to the same customer) can be batched together, and a common processing time occurs for the batch. Jobs can also have different priorities (weights). Finally, all jobs have different sizes, and the batching machine is capacitated. The objective is to find a schedule that minimises TWC time of the jobs. By assuming the release times are equal to 0 and there is only a single machine, the problem reduces to the batch scheduling problem proven to be NP-hard by Dobson and Nambimadom (2001). Since the problem is NP-hard, we present different heuristic approaches in the following section. The proposed optimisation model is used to evaluate the effectiveness of heuristics for small problem instances.

4. Heuristics

Given the problem's complexity, heuristic algorithms are needed to solve any practical size problem instances. Two important decisions to be made are batch formation and batch sequencing. The problem also involves machine assignment decisions. In our approach, there is a construction phase followed by an improvement phase. The constructive heuristic presented is based on common approaches found in the scheduling literature such as WSPT (weighted shortest processing time first) rule to minimise TWC time on a single machine. Then, in the improvement phase, different neighbourhood search techniques are utilised to produce better solutions.

Variable neighbourhood search (VNS) is a meta-heuristic approach that explores the solution space by allowing the change of neighbourhoods (Mladenovic and Hansen 1997). Many successful applications are discussed in the survey paper on VNS by Hansen and Mladenovic (2001). VNS combines a systematic change of neighbourhoods with a local search procedure. First, a random solution is selected from the initial neighbourhood. Then, a local search is performed to find a local optimal solution. If the best solution found so far is improved, the procedure is restarted with the first neighbourhood; otherwise, it proceeds with the next neighbourhood. The stopping criteria can be defined by a limit on total computational time, total number of iterations, number of iterations without any improvement, or computational time without any improvement.

Different local search procedures based on insertion and/or swap moves of the jobs and/or batches are examined for our problem. It is expected that the quality of the solution increases if the size of the neighbourhood increases (Agarwal et al. in press). In our case of VNS, neighbourhood structure is associated with a distance measure. Size of the neighbourhood increases when insertion and/or swap moves are allowed within jobs or batches positioned further afield based on their processing start times. Job insertions and swaps are performed within the pre-determined positions of the corresponding family's batches. For example, if the neighbourhood structure has a distance of one (denoted N_1), a job can only be inserted into another batch of the same family that is adjacent according to the position of batches within the corresponding family. (In our example problem schedule shown in Figure 2, job 6 can only be inserted into the trip including jobs 3 and 5. It cannot be inserted into the trip with job 1 because trips are not adjacent.) For job moves, batch positions are the order of the batches of a particular family and determined based on the starting time of batches regardless of the assigned machine. Batches are only ordered among the other batches of the corresponding family. Ties are broken arbitrarily. Similarly, batch insertions and swaps are also done within pre-determined batch positions. In this case, overall batch positions among all batches are considered instead of positions within associated families. Again, positions are determined based on the processing start times of the batches regardless of the assigned machine. Therefore, all neighbourhood definitions include different machines as well as the same machine that the job or batch is assigned.

Step 1. Intialisation. Define neighbourhood structures N_k $(k=1,...,k_{max})$, find an initial solution S^*, and set $k=1$.
Step 2. Local search. Perform local search and find the best solution S' on $N_k(S^*)$.
Step 3. Move or not. If $G(S') < G(S^*)$, where G(^) is *TWC* of ^, then $S^* \leftarrow S'$; $k=1$; go to Step 2. Otherwise, go to Step 4.
Step 4. Neighbourhood update. $k \leftarrow k+1$. If $k > k_{max}$, stop. Otherwise, go to Step 2.

Figure 3. Proposed VND algorithm.

After an initial solution is found and neighbourhood distance is set as 1, a local search is performed until the stopping criteria is reached, which is defined as the number of iterations (n_iter). In VNS, every time before performing local search, a random solution is selected from the defined neighbourhood of the best solution so far. Because release times and incompatible job families exist in our problem, achieving good solutions by starting the search from a random solution is quite difficult. Therefore, a variant of VNS, Variable Neighbourhood Descent (VND), is applied in this study. In VND, the neighbourhood is explored without finding a random initial solution before a local search. The remaining steps are same as VNS. Figure 3 details the proposed algorithm. Once the best solution found so far (S^*) is improved, the procedure is restarted with the first neighbourhood. If the best solution found in the last local search does not better than S^*, algorithm explores the next neighbourhood in local search. And, algorithm stops when no improvement is achieved in all neighbourhood searches.

In order to explore neighbourhoods effectively, we investigated different local search procedures, both separately and sequentially. The first two procedures are based on the insertions and swaps of the jobs (Figures 4 and 5). A job insert move removes a job from one batch and inserts it into another. A job swap move selects two jobs from the same family and switches their batch assignments. Swap and insert moves of the batches are also examined in local search procedures (Figures 6 and 7). Job swap moves always yield the same number of jobs assigned to batches and machines. Similarly, the number of batches processed on each machine remains constant when batch swaps are applied. On the other hand, any improvement involving more than a single job's batch re-assignment or a single batch's repositioning is not easily found with an insert move. Therefore, local search procedures are also applied together in a sequential manner to overcome these weaknesses. For example, one algorithm can first apply job insert moves and continue to explore the neighbourhood with job swaps (Figure 8).

Starting solutions generally have significant impact on the solution quality of algorithms, especially in neighbourhood search algorithms (Naderi et al. 2008). Therefore, in order to find good initial solutions, simple dispatching rules are applied for assigning jobs to batches and sequencing these batches in our study. Many problem characteristics such as weight, volume, release, and processing times are taken into consideration in these dispatching rules. First, jobs are sorted and assigned to first available batch based on these parameters. Then, batches are sorted and scheduled on first idle machine. This constructive heuristic approach is used to build initial solutions for our neighbourhood search algorithms rather than starting them from random solutions. The five-step approach to build both batch formations and schedules is shown in Figure 9. In Figure 9, Ψ is used to aggregate lateness and job size in the denominator such that varying its value between 0 and 1 can blend these two factors with varying levels of importance. Therefore, different values of Ψ can help to further explore the solution space.

Although smaller batches can lead to earlier completion times, hence a smaller objective, this does not imply that there should never be an idle machine waiting for the next job to be added to a non-empty batch. Consider a simple example with two jobs and one batching machine in the system. Jobs have the same weight, volume, and family. Job 1 is released at time 0 and job 2 is released at time 5. Machine loading capacity is big enough to process both jobs simultaneously and processing time is 12. If the machine is never kept idle and job 1 is processed first followed by job 2, resulting schedule (A1) has an objective value of $12\omega+24\omega = 36\omega$ in which ω is the weight for each job. In an alternative schedule (A2), a better objective value can be achieved by processing jobs simultaneously. The machine is idle until job 2 is released and objective value is $(5+12)\omega + (5+12)\omega = 34\omega$. Both schedules are illustrated in Figure 10.

Based on our finding explained above, an accept–reject rule can be adopted in Step 2 of the proposed constructive heuristic while allocating jobs to batches. In the existence of different job release times, the idea of delaying the start of a batch in order to accommodate jobs arriving soon when it is advantageous from the total system performance

$n = 1$
$S' = S^*$
While $n < n\_iter$ Do
  Randomly select job i from the available jobs
  Remove job i from the current batch assignment
  Select a random batch y on $N_k(S^*)$
  Insert i into batch y and get a new schedule S
  If $G(S) < G(S')$ and S is feasible then $S' = S$
  $n = n + 1$
End While

Figure 4. Local search 1 (LS1): job insert.

$n = 1$
$S' = S^*$
While $n < n\_iter$ Do
  Randomly select job i and job j respectively from batch y and batch z on $N_k(S^*)$
  Assign job i to batch z and assign job j to batch y to obtain a new schedule S
  If $G(S) < G(S')$ and S is feasible then $S' = S$
  $n = n + 1$
End While

Figure 5. Local search 2 (LS2): job swap.

$n = 1$
$S' = S^*$
While $n < n\_iter$ Do
  Randomly select batch y from the available batches. Remove batch y from the current position
  Insert it to a random position p on $N_{2k}(S^*)$ and get a new schedule S
  If $G(S) < G(S')$ and S is feasible then $S' = S$
  $n = n + 1$
End While

Figure 6. Local search 3 (LS3): batch insert.

$n = 1$
$S' = S^*$
While $n < n\_iter$ Do
  Randomly select batch y and batch z from $N_{2k}(S^*)$
  Swap positions of y and z and get a new schedule S
  If $G(S) < G(S')$ and S is feasible then $S' = S$
  $n = n + 1$
End While

Figure 7. Local search 4 (LS4): batch swap.

standpoint is effectively used in Fowler, Phillips, and Hogg (1992) and Lee and Uzsoy (1999). The trade-off between total delay caused in the current batch by waiting for the next job and possible delay eliminated in the subsequent jobs is evaluated. Information on future job arrivals and batch processing times are used with some user-specified parameters in order to determine whether processing should be started or postponed. Although sending large jobs first may help minimise the number of batches (i.e. bin-packing), our objective of minimising total weighted completion time drives us to select smaller jobs first so that we can send more jobs to customers quickly. Our approach is more similar to the extended greedy ratio (EXGR) heuristic of Uzsoy (1994). Before adding each job to the batch, it is ensured that batch quality is improved, which is defined as the index used to sort batches in Step 3 of the constructive heuristic. For that reason, step 2 is modified as follows by applying an accept/reject check:

Step 2 – modified: Jobs are sorted in non-increasing order of $I_j(t)$ and added to the first available batch of the corresponding family wherein the batch sequencing index in Step 3 is improved by adding the job to the batch.

5. Experimental study

In this study, we compare different local search procedures within a VNS scheme. When employing the construction phase, constant values of Step 1 are investigated in 0.1 increments of Ψ starting from 0.1 such that

$n = 1$
$S' = S^*$
While $n < n\_iter$ Do
  Randomly select job i from the available jobs
  Remove job i from the current batch assignment
  Select a random batch y on $N_k(S^*)$
  Insert i into batch y and get a new schedule S
  If $G(S) < G(S')$ and S is feasible then $S' = S$
  $n = n + 1$
End While
If $S' < S^*$ then $S^* = S'$ End If
$n = 1$
While $n < n\_iter$ Do
  Randomly select job i and job j respectively from batch y and batch z on $N_k(S^*)$
  Assign job i to batch z and assign job j to batch y to obtain a new schedule S
  If $G(S) < G(S')$ and S is feasible then $S' = S$
  $n = n + 1$
End While

Figure 8. LS1 + LS2.

Step 1: Whenever a machine becomes idle, a ranking index $I_j(t)$ is calculated for each unscheduled job $j \in J$ at time t: $I_j(t) = \left\{ \frac{w_j}{\Psi v_j + (1-\Psi)\max\{0, r_j - t\}} \right\}$, where Ψ is a constant value.

Step 2: Jobs are sorted in non-increasing order $I_j(t)$. Then, jobs are added to first available batch of the corresponding family.

Step 3: The batch with the highest $\omega_k / (\max\{0, \Re_k - t\} + P_k)$ value is selected to be scheduled on the idle machine at time t, where,

- P_k is the processing time required for jobs in the batch k.
- $\Re_k$ is the maximum release time of jobs in the batch k.
- ω_k is the total weight of the jobs included in the batch k.

Step 4: t must be updated to the earliest point in time at which conditions change. Conditions change when (a) the selected batch finishes its processing and (b) another machine becomes idle.

Step 5: If there is no unscheduled job, stop. Otherwise, go to Step 1.

Figure 9. Constructive heuristic.

($\Psi = 0.1,\ \Psi = 0.2, \cdots, \Psi = 1$). The constant value of 0 is not chosen because it results in an undefined sorting index value for released jobs (i.e. divides by zero). The best solution achieved with any of these combinations of the constant values is selected to be the initial solution. Maximum neighbourhood size (k_{max}) is set to 3, and the number of iterations performed by each local search procedure (n_iter) is set as 100. Different heuristic algorithms are tested by applying diverse combinations of proposed local search procedures and the accept/reject check (Table 1).

A set of computational experiments is designed to evaluate the performance of each heuristic. Different characteristics of the problem are taken into account for generating random problem instances. We consider four levels of the

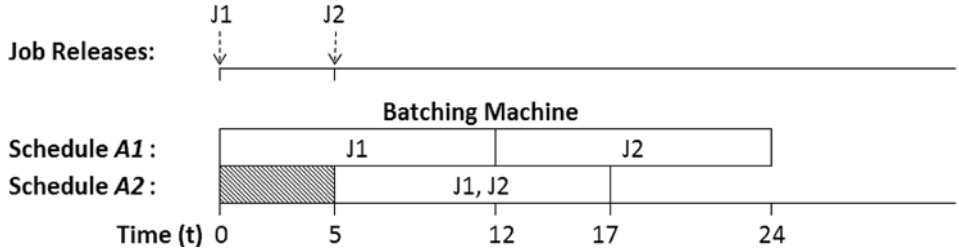

Figure 10. Alternative schedules.

Table 1. Heuristic descriptions.

Heuristic	Apply accept/reject check?	Local search
MH1	No	None
MH2	Yes	None
MH3	No	LS1
MH4	Yes	LS1
MH5	No	LS2
MH6	Yes	LS2
MH7	No	LS1 + LS2
MH8	Yes	LS1 + LS2
MH9	No	LS2 + LS1
MH10	Yes	LS2 + LS1
MH11	No	LS3
MH12	Yes	LS3
MH13	No	LS4
MH14	Yes	LS4
MH15	No	LS3 + LS4
MH16	Yes	LS3 + LS4
MH17	No	LS4 + LS3
MH18	Yes	LS4 + LS3
MH19	No	LS2 + LS1 + LS4 + LS3
MH20	Yes	LS2 + LS1 + LS4 + LS3
MH21	No	LS4 + LS3 + LS2 + LS1
MH22	Yes	LS4 + LS3 + LS2 + LS1

number of jobs: 15, 25, 50, and 100. Each job is assigned to exactly one of the families. Two different levels of number of job families are investigated: 3 and 5. Batch capacity (δ) is set as 50, and two different levels of job sizes are generated from a discrete uniform distribution by using different ranges. Ranges for the two levels are [1, 25] and [1, 50]. Experiments are performed for both two and three machines operating in parallel. The processing times for each batch associated with a specific family are randomly assigned as discrete integer values on two uniform distributions: [1, 15] and [6, 10]. Both distributions have the same mean but different variance. After specifying processing times for a given instance, release times are generated similar to Malve and Uzsoy (2007). A random number is generated on the interval [0,v], $v \in \{0.5, 1\}$, and multiplied by a lower bound estimation for the makespan, $C_{\max LB}$. $C_{\max LB}$ is calculated as $\frac{\sum_{f \epsilon F} \beta_f \left\lceil \frac{v_f}{\delta} \right\rceil}{m}$, in which β_f is the processing time required for family f, and v_f is the total size of the jobs associated with family f. Finally, two levels of job weights are randomly generated from a discrete integer uniform distribution ranging from 1 to 5 and 1 to 10. For each combination of the levels, 10 problem instances are generated yielding a total of 2560 ($10 \times 4 \times 2^6$) problems. Table 2 summarises the experimental design.

6. Experimental results

We perform two types of comparisons in order to test the effectiveness of the proposed solution approaches. First, we compare the best solution achieved by any of the heuristics with the optimal solutions in small size (15 jobs) instances. Then, each heuristic's individual performance across all instances is evaluated against the best heuristic performance due

Table 2. Experimental design.

Factors	Levels	Level description
Number of machines	2	2, 3
Number of jobs	4	15, 25, 50, 100
Number of families	2	3, 5
Job release times	2	$U(0,\alpha)C_{\max LB}$: α = 0.5 or 1
Batch processing times	2	DU[6,10], DU[1,15]
Job weights	2	DU[1,5], DU[1,10]
Batch capacity	1	50
Job sizes	2	DU[1,50], DU[1,25]
	256	

to the problem's NP hard complexity. The heuristic algorithms are implemented in Visual Basic for Applications (Excel 2007), and the mathematical model is solved using IBM ILOG CPLEX 11.1. Tests are carried out on a PC with an Intel Pentium dual-core processor (3.39 GHz CPU speed) and 3 GB of RAM. Computational time differences within heuristics were negligible and 15 jobs were the biggest problem size that we were able to get optimal solution in a couple of hours for all number of machines and job families.

Let $TWCT(O)$ be the optimal objective value obtained via the mathematical program and $TWCT(B^*)$ be the best value obtained from any of the heuristics. The performance ratio, $PR(B^*) = \frac{TWCT(B^*)}{TWCT(O)}$, is computed to assess the solution quality of the best heuristic in small size instances. In Table 3 we summarise the results for these instances. The best heuristic is capable of providing near-optimal solutions in the 15-job problem instances. Then, another performance ratio, $PR(MH) = \frac{TWCT(MH)}{TWCT(B^*)}$, is computed to assess the solution quality of the different heuristics in all instances, where $TWCT(MH)$ is the total weighted completion time value achieved by applying each of the heuristics listed in Table 1. The average performance ratio value computed via the best heuristic for each experimental factor combination is given in Table 4. The number of times that a given heuristic produced the best overall solution for a problem instance at each factor level is reported in parentheses, and best performances are shown in bold.

The accept–reject rule appears to be quite promising. The heuristics for which the rule is applicable are further evaluated to see the impact on performance (Table 5). For example, H2 improves significantly upon H1 in terms of average performance ratio. On average, a 12.1% improvement is achieved by applying the accept–reject rule before assigning jobs to the batches with available capacity. Among local searches, repositioning the batches instead of jobs yields better results. As batches typically contain more than one job, the search neighbourhood associated with batch repositioning parallels a k-exchange, where k denotes the number of jobs in a given batch. Therefore, we conjecture that this modified neighbourhood affords greater improvement in objective function value than does a single job move (i.e. 1-exchange).MH12 (batch inserts only) outperforms the other local searches when they are applied individually.

Table 3. Best heuristic performance compared to optimality.

Factor	Level	Avg *PR*
Number of families	3	1.033
	5	1.028
Number of machines	2	1.033
	3	1.028
Job processing times	DU[6,10]	1.030
	DU[1,15]	1.031
Job release times (constants)	0.5	1.024
	1	1.038
Job sizes	DU[1,25]	1.038
	DU[1,50]	1.023
Job weights	DU[1,5]	1.030
	DU[1,10]	1.031
	Overall	**1.031**

Table 4. Heuristic results: average performance ratio (number of best solutions).

	Number of Jobs				Number of Families		Number of Machines		Job Processing Times		Job Sizes		Job Release Times (constants)		Job Weights		
Heuristic	15	25	50	100	3	5	2	3	DU[6,10]	DU[1,15]	DU[1,25]	DU[1,50]	0.5	1	DU[1,5]	DU[1,10]	**Overall**
MH1	1.133 (34)	1.182 (7)	1.227 (0)	1.236 (0)	1.199 (26)	1.190 (15)	1.207 (21)	1.182 (20)	1.191 (30)	1.198 (11)	1.193 (34)	1.196 (7)	1.160 (34)	1.229 (7)	1.193 (26)	1.196 (15)	1.195 (41)
MH2	1.058 (99)	1.066 (49)	1.047 (17)	1.026 (3)	1.051 (97)	1.048 (71)	1.046 (97)	1.052 (71)	1.047 (100)	1.052 (68)	1.032 (148)	1.066 (20)	1.048 (119)	1.051 (49)	1.050 (97)	1.049 (71)	1.049 (168)
MH3	1.092 (60)	1.119 (18)	1.168 (0)	1.202 (0)	1.144 (50)	1.146 (28)	1.159 (31)	1.132 (47)	1.143 (51)	1.147 (27)	1.139 (66)	1.151 (12)	1.119 (57)	1.171 (21)	1.144 (45)	1.147 (33)	1.145 (78)
MH4	1.046 (125)	1.052 (70)	1.040 (28)	1.023 (3)	1.041 (134)	1.040 (92)	1.040 (117)	1.041 (109)	1.039 (132)	1.042 (94)	1.023 (197)	1.058 (29)	1.040 (148)	1.041 (78)	1.041 (131)	1.040 (95)	1.041 (226)
MH5	1.130 (34)	1.177 (9)	1.222 (0)	1.233 (0)	1.194 (28)	1.187 (15)	1.203 (21)	1.178 (22)	1.188 (32)	1.194 (11)	1.189 (36)	1.192 (7)	1.157 (36)	1.225 (7)	1.189 (26)	1.192 (17)	1.191 (43)
MH6	1.056 (103)	1.063 (54)	1.045 (21)	1.025 (5)	1.048 (103)	1.046 (80)	1.045 (102)	1.050 (81)	1.045 (111)	1.050 (72)	1.031 (159)	1.064 (24)	1.046 (132)	1.049 (51)	1.048 (105)	1.047 (78)	1.047 (183)
MH7	1.089 (63)	1.114 (24)	1.157 (0)	1.192 (0)	1.136 (59)	1.140 (28)	1.152 (32)	1.125 (55)	1.136 (57)	1.140 (30)	1.133 (75)	1.143 (12)	1.114 (65)	1.162 (22)	1.136 (48)	1.140 (39)	1.138 (87)
MH8	1.044 (131)	1.050 (78)	1.038 (32)	1.022 (10)	1.038 (149)	1.039 (102)	1.038 (132)	1.039 (119)	1.037 (150)	1.040 (101)	1.022 (216)	1.055 (35)	1.037 (167)	1.039 (84)	1.039 (144)	1.038 (107)	1.038 (251)
MH9	1.089 (64)	1.113 (23)	1.158 (2)	1.192 (0)	1.135 (61)	1.141 (28)	1.152 (34)	1.124 (55)	1.136 (58)	1.140 (31)	1.133 (75)	1.143 (14)	1.114 (67)	1.163 (22)	1.137 (49)	1.139 (40)	1.138 (89)
MH10	1.044 (131)	1.050 (80)	1.038 (39)	1.022 (13)	1.038 (156)	1.039 (107)	1.038 (135)	1.039 (128)	1.037 (159)	1.040 (104)	1.022 (227)	1.055 (36)	1.037 (175)	1.039 (88)	1.038 (149)	1.038 (114)	1.038 (263)
MH11	1.078 (104)	1.116 (23)	1.172 (1)	1.201 (1)	1.146 (61)	1.138 (68)	1.148 (73)	1.136 (56)	1.140 (78)	1.144 (51)	1.157 (64)	1.127 (65)	1.113 (101)	1.171 (28)	1.142 (71)	1.142 (58)	1.142 (129)
MH12	1.018 (313)	1.029 (161)	1.021 (99)	1.013 (44)	1.022 (273)	1.018 (344)	1.017 (363)	1.023 (254)	1.019 (329)	1.021 (288)	1.016 (385)	1.024 (232)	1.021 (354)	1.020 (263)	1.021 (317)	1.019 (300)	1.020 (617)
MH13	1.084 (93)	1.124 (15)	1.176 (1)	1.203 (0)	1.151 (48)	1.142 (61)	1.155 (58)	1.139 (51)	1.144 (71)	1.149 (38)	1.161 (62)	1.133 (47)	1.115 (86)	1.178 (23)	1.147 (63)	1.147 (46)	1.147 (109)
MH14	1.023 (256)	1.033 (149)	1.023 (76)	1.013 (33)	1.026 (219)	1.021 (295)	1.021 (288)	1.025 (226)	1.023 (286)	1.024 (228)	1.018 (358)	1.028 (156)	1.023 (305)	1.023 (209)	1.023 (272)	1.023 (242)	1.023 (514)
MH15	1.076 (112)	1.113 (26)	1.165 (2)	1.195 (0)	1.142 (63)	1.133 (77)	1.143 (76)	1.132 (64)	1.135 (87)	1.140 (53)	1.154 (65)	1.120 (75)	1.107 (108)	1.167 (32)	1.137 (81)	1.138 (59)	1.137 (140)
MH16	1.017 (323)	1.025 (186)	1.018 (122)	1.011 (58)	1.020 (305)	1.016 (384)	1.015 (397)	1.020 (292)	1.017 (372)	1.019 (317)	1.015 (401)	1.020 (288)	1.018 (393)	1.018 (296)	1.018 (358)	1.017 (331)	1.018 (689)
MH17	1.077 (111)	1.114 (25)	1.166 (2)	1.194 (0)	1.141 (64)	1.133 (74)	1.144 (79)	1.131 (59)	1.135 (85)	1.140 (53)	1.154 (65)	1.121 (73)	1.108 (107)	1.167 (31)	1.137 (79)	1.138 (59)	1.137 (138)
MH18	1.017 (316)	1.026 (183)	1.018 (111)	1.010 (70)	1.020 (300)	1.016 (380)	1.015 (382)	1.020 (298)	1.017 (360)	1.019 (320)	1.015 (398)	1.020 (282)	1.018 (388)	1.018 (292)	1.018 (348)	1.017 (332)	1.018 (680)
MH19	1.037 (295)	1.050 (150)	1.088 (41)	1.139 (10)	1.079 (260)	1.078 (236)	1.086 (229)	1.070 (267)	1.078 (278)	1.078 (218)	1.093 (233)	1.064 (263)	1.061 (313)	1.095 (183)	1.078 (257)	1.079 (239)	1.078 (496)
MH20	**1.004 (533)**	**1.009 (420)**	**1.009 (352)**	1.005 (284)	1.007 (775)	**1.006 (814)**	**1.007 (808)**	1.006 (781)	**1.006 (819)**	1.007 (770)	1.004 (912)	**1.009 (677)**	1.007 (785)	**1.005 (804)**	1.006 (793)	**1.006 (796)**	**1.006 (1589)**
MH21	1.040 (291)	1.052 (155)	1.092 (31)	1.136 (9)	1.079 (256)	1.081 (230)	1.089 (231)	1.071 (255)	1.080 (273)	1.080 (213)	1.096 (228)	1.064 (258)	1.063 (312)	1.097 (174)	1.080 (237)	1.080 (249)	1.080 (486)
MH22	1.004 (527)	1.009 (413)	1.009 (338)	**1.004 (315)**	**1.007 (792)**	1.006 (801)	1.007 (789)	**1.006 (804)**	1.006 (813)	**1.007 (780)**	**1.003 (936)**	1.010 (657)	**1.007 (787)**	1.006 (806)	**1.006 (825)**	1.007 (768)	1.007 (1593)

Table 5. Performance ratio results with/without the accept–reject rule.

	Without accept–reject	With accept–reject
MH1 vs. MH2	1.195	1.049
MH3 vs. MH4	1.145	1.041
MH5 vs. MH6	1.191	1.047
MH7 vs. MH8	1.138	1.038
MH9 vs. MH10	1.138	1.038
MH11 vs. MH12	1.142	1.020
MH13 vs. MH14	1.147	1.023
MH15 vs. MH16	1.137	1.018
MH17 vs. MH18	1.137	1.018
MH19 vs. MH20	1.078	1.006
MH21 vs. MH22	1.080	1.007
	1.139	1.028

Employing different local searches sequentially within the same VND procedure consistently performs well (Table 6). However, the ordering of the local searches does not particularly show any significant impact on solution qualities (i.e. MH8 vs. MH10 or MH16 vs. MH18). The solution quality is improved when parameters cause less compatible job combinations for available batches therefore less candidates for insert-swap moves (i.e. more number of families and bigger job sizes). On the other side, relaxations through more machines and closer release times appear to make it easier for our heuristics to better allocate jobs to available batches initially and then perform improvement moves.

We also evaluate the variance of performance as discussed in Hall and Posner (2007) through confidence intervals. Table 6 illustrates the 95% confidence intervals for $PR(MH)$ for all proposed heuristic solution approaches, sorted according to non-decreasing lower confidence limit (LCL). Best heuristic performances which are statistically indistinguishable from each other (i.e. those with overlapping of the confidence intervals) are shown in bold. Heuristics MH20 and MH22 are the best, and their intervals do not overlap with other heuristics' confidence intervals. Both heuristics use all local searches sequentially with an accept–reject evaluation. In order to determine any significance difference between performances of MH20 and MH22, a paired t-test at 95% degree of confidence (Table 7) is employed. Null and alternative hypothesis are defined as H_0: $\mu_{PR(MH20)} - \mu_{PR(MH22)} = 0$ and H_1: $\mu_{PR(MH20)} - \mu_{PR(MH22)} \neq 0$, respectively. Through t-test results, it is shown that at 95% confidence there is not a significant difference between performances of MH20 and MH22.

The performance of heuristics are further evaluated by calculating performance ratios after pre-specified time breaks (Table 8). Computational experiments indicate that MH20 (in which local searches are applied in sequence of job swaps, job inserts, batch swaps, and batch inserts) has excellent average performance with a reasonable computational burden. Extensive computational experiments show that only in the first 10 seconds did MH12 (includes only batch insert) converge to the best solution better than MH20, providing a good trade-off between solution time and quality. Accordingly, although MH10 and MH20 both include job swaps and inserts, MH20 performs much better than MH10 within 10 seconds. This is reasonable since MH20 also includes batch swaps and inserts. As computational time increases, MH20 converges to best solutions much faster and finds the best solution in 1,589 out of the 2,560 problem instances. From these results, it is clear that employing different local searches sequentially under a VNS scheme is an effective strategy for minimising *TWC* in parallel batch machine scheduling problems with incompatible job families and job release times.

7. Conclusions and future research

In this paper, we study the problem of minimising weighted completion time on batching machines with incompatible job families. This study introduces a supply-chain scheduling problem as a batch machine scheduling problem. In addition to considering unequal job sizes, our research expands on existing assumptions described in the literature by involving different job release times and parallel batching machines. We present a MIP formulation that can optimally solve small problem instances. We then propose and show the efficiency of different local search algorithms individually and sequentially under a variable neighbourhood search scheme.

A constructive heuristic is used to find good starting solutions for our neighbourhood search algorithms. The proposed accept–reject rule has been shown to be very effective for improving the quality of batch formations in the

Table 6. 95% Confidence intervals of performance ratios for heuristics.

15 jobs			25 jobs			50 jobs			100 jobs		
Heuristic	LCL	UCL	Heuristic	LCL	UCL	Heuristic	LCL	UCL	Heuristic	LCL	UCL
MH20	**1.003**	**1.004**	**MH20**	**1.007**	**1.010**	**MH20**	**1.007**	**1.010**	**MH22**	**1.004**	**1.005**
MH22	**1.003**	**1.005**	**MH22**	**1.008**	**1.011**	**MH22**	**1.007**	**1.010**	**MH20**	**1.004**	**1.005**
MH16	1.015	1.019	MH16	1.023	1.028	MH16	1.016	1.020	MH18	1.009	1.011
MH18	1.015	1.019	MH18	1.023	1.028	MH18	1.016	1.020	MH16	1.010	1.012
MH12	1.016	1.020	MH12	1.026	1.031	MH12	1.019	1.023	MH12	1.012	1.014
MH14	1.021	1.025	MH14	1.030	1.036	MH14	1.021	1.025	MH14	1.012	1.015
MH19	1.033	1.041	MH19	1.045	1.054	MH10	1.035	1.041	MH10	1.020	1.023
MH21	1.035	1.044	MH10	1.046	1.053	MH8	1.035	1.041	MH8	1.021	1.023
MH8	1.041	1.047	MH8	1.046	1.053	MH4	1.038	1.043	MH4	1.022	1.025
MH10	1.041	1.047	MH21	1.047	1.056	MH6	1.042	1.048	MH6	1.023	1.026
MH4	1.043	1.049	MH4	1.049	1.056	MH2	1.044	1.050	MH2	1.024	1.027
MH6	1.052	1.059	MH6	1.059	1.068	MH19	1.083	1.094	MH21	1.130	1.142
MH2	1.054	1.062	MH2	1.062	1.070	MH21	1.086	1.097	MH19	1.133	1.145
MH15	1.071	1.082	MH15	1.107	1.119	MH7	1.151	1.163	MH7	1.186	1.198
MH17	1.071	1.082	MH9	1.107	1.119	MH9	1.152	1.165	MH9	1.186	1.198
MH11	1.073	1.084	MH17	1.108	1.120	MH15	1.158	1.172	MH17	1.187	1.200
MH13	1.078	1.090	MH7	1.108	1.120	MH17	1.159	1.172	MH15	1.188	1.201
MH9	1.083	1.095	MH11	1.110	1.123	MH3	1.161	1.174	MH11	1.194	1.207
MH7	1.084	1.095	MH3	1.113	1.125	MH11	1.165	1.179	MH3	1.196	1.208
MH3	1.087	1.098	MH13	1.118	1.131	MH13	1.169	1.183	MH13	1.197	1.210
MH5	1.123	1.137	MH5	1.170	1.185	MH5	1.214	1.230	MH5	1.227	1.240
MH1	1.126	1.140	MH1	1.174	1.190	MH1	1.219	1.235	MH1	1.230	1.243

Table 7. Paired t-test results of MH20 vs. MH22 with $\alpha = 0.05$.

No. of jobs, no. of machines	$\mu_{PR(MH20)}$	$\mu_{PR(MH22)}$	t-value	p-value	Reject H_0 ?
15, 2	1.0044	1.0053	–0.761	0.447	No
15, 3	1.0027	1.0036	–1.194	0.233	No
25, 2	1.0094	1.0091	0.180	0.857	No
25, 3	1.0081	1.0091	–0.725	0.469	No
50, 2	1.0081	1.0080	0.053	0.958	No
50, 3	1.0090	1.0091	–0.021	0.983	No
100, 2	1.0046	1.0043	0.398	0.691	No
100, 3	1.0048	1.0042	0.965	0.335	No

Table 8. Heuristics performance over time.

Heuristic	After 10 s	After 30 s	After 60 s	After 120 s	No time limit
MH1	1.198	1.195	1.195	1.195	1.195
MH2	1.052	1.049	1.049	1.049	1.049
MH3	1.146	1.145	1.145	1.145	1.145
MH4	1.041	1.041	1.041	1.041	1.041
MH5	1.190	1.191	1.191	1.191	1.191
MH6	1.049	1.047	1.047	1.047	1.047
MH7	1.161	1.150	1.145	1.141	1.138
MH8	1.042	1.039	1.039	1.038	1.038
MH9	1.159	1.152	1.146	1.142	1.138
MH10	1.044	1.039	1.039	1.038	1.038
MH11	1.150	1.144	1.143	1.142	1.142
MH12	**1.025**	1.022	1.021	1.020	1.020
MH13	1.159	1.152	1.150	1.148	1.147
MH14	1.028	1.025	1.024	1.023	1.023
MH15	1.157	1.147	1.142	1.140	1.137
MH16	1.026	1.021	1.019	1.018	1.018
MH17	1.157	1.147	1.143	1.140	1.137
MH18	1.026	1.021	1.020	1.018	1.018
MH19	1.138	1.124	1.109	1.097	1.078
MH20	1.028	**1.018**	**1.013**	**1.010**	**1.006**
MH21	1.155	1.127	1.113	1.101	1.080
MH22	1.028	1.018	1.013	1.010	1.007

construction phase. The performances of the heuristics are further improved by using VND algorithms. We have shown that among local searches, repositioning the batches instead of jobs yields better results. Batch insertion moves outperform other local searches when they are applied individually. Extensive computational tests demonstrate that there is a distinct advantage in using different local search algorithms together in a sequential manner as compared with employing them individually.

There are several possible extensions to our research. One is to study due-date-related objective performance measures, such as total weighted tardiness, along with the corresponding changes to the experimental design to investigate different levels of due-date range and due-date tightness. Another direction for future research is to consider job-shop and flow-shop environments involving batching machines. It would also be interesting to include cost functions in the objective such as customer waiting time cost (cost of *TWC*) and cost of more machines (trucks) in which the number of machines is a decision variable. Furthermore, multi-objective batch scheduling problems can be investigated in future research.

References

Agarwal, R. et al. in press. “Solving parallel machine scheduling problems with variable depth local search”. *Journal of Scheduling*.

Azizoglu, M., and S. Webster. 2001. “Scheduling a batch processing machine with incompatible job families.” *Computers and Industrial Engineering* 39: 325–335.

Balasubramanian, H., et al. 2004. “Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness.” *International Journal of Production Research* 42: 1621–1638.

Cheng, T. C. E., and M. Y. Kovalyov. 2001. “Single machine batch scheduling with sequential job processing.” *IIE Transactions* 33: 413–429.

Dobson, G., and R. S. Nambimadom. 2001. “The batch loading and scheduling problem.” *Operations Research* 49: 52–65.

Erramilli, V., and S. J. Mason. 2008. “Multiple orders per job batch scheduling with incompatible job families.” *Annals of Operations Research* 159: 245–260.

Fowler, J. W., D. T. Phillips, and G. L. Hogg. 1992. “Real-time control of multiproduct bulk-service semiconductor manufacturing processes.” *IEEE Transactions on Semiconductor Manufacturing* 5: 158–163.

Hall, N. G., and M. E. Posner. 2007. “Performance prediction and preselection for optimization and heuristic solution procedures.” *Operations Research* 55: 703–716.

Hansen, P., and N. Mladenovic. 2001. “Variable neighborhood search: principles and applications.” *European Journal of Operational Research* 130: 449–467.

Keha, A. B., K. Khowala, and J. W. Fowler. 2009. “Mixed integer programming formulations for single machine scheduling problems.” *Computers and Industrial Engineering* 56: 357–367.

Koh, S. G., et al. 2005. “Scheduling a single batch processing machine with arbitrary job sizes and incompatible job families.” *International Journal of Production Economics* 98: 81–96.

Kurz, M. E., and S. J. Mason. 2008. “Minimizing total weighted tardiness on a batch-processing machine with incompatible job families and job ready times.” *International Journal of Production Research* 46: 131–151.

Lee, C. Y., and R. Uzsoy. 1999. “Minimizing makespan on a single batch processing machine with dynamic job arrivals.” *International Journal of Production Research* 37: 219–236.

Malve, S., and R. Uzsoy. 2007. “A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families.” *Computers and Operations Research* 34: 3016–3024.

Mladenovic, N., and P. Hansen. 1997. “Variable neighborhood search.” *Computers and Operations Research* 24: 1097–1100.

Mönch, L., et al. 2005. “Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times.” *Computers and Operations Research* 32: 2731–2750.

Naderi, B., et al. 2008. “A variable neighborhood search for hybrid flexible flowshops with setup times minimizing total completion time.” *Journal of Applied Sciences* 8: 2843–2850.

Perez, I. C., J. W. Fowler, and W. M. Carlyle. 2005. “Minimizing total weighted tardiness on a single batch process machine with incompatible job families.” *Computers and Operations Research* 32: 327–341.

Pinedo, M. 2012. *Scheduling, Theory, Algorithms, and Systems*. 4th ed. New York: Springer.

Potts, C. N., and M. Y. Kovalyov. 2000. “Scheduling with batching: a review.” *European Journal of Operational Research* 120: 228–249.

Sung, C. S., et al. 2002. “Minimizing makespan on a single burn-in oven with job families and dynamic job arrivals.” *Computers and Operations Research* 29: 995–1007.

Unlu, Y., and S. J. Mason. 2010. “Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems.” *Computers and Industrial Engineering* 58: 785–800.

Uzsoy, R. 1994. “Scheduling a single batch processing machine with non-identical job sizes.” *International Journal of Production Research* 32: 1615–1635.

Uzsoy, R. 1995. “Scheduling batch processing machines with incompatible job families.” *International Journal of Production Research* 33: 2685–2708.