Chapter 12
Batching-Based Approaches for Optimized Packing of Jobs in the Spatial Scheduling Problem

Sudharshana Srinivasan, J. Paul Brooks, and Jill Hardin Wilson

Abstract Spatial resources are often an important consideration in shipbuilding and large-scale manufacturing industries. Spatial scheduling problems (SSP) involve the non-overlapping arrangement of jobs within a limited physical workspace such that some scheduling objective is optimized. The jobs are typically heavy and occupy large areas, requiring that the same contiguous units of space be assigned throughout the duration of their processing time. This adds an additional level of complexity to the general scheduling problem. Since solving large instances using exact methods becomes computationally intractable, there is a need to develop alternate solution methodologies to provide near optimal solutions for these problems. Much of the literature focuses on minimizing the makespan of the schedule. We propose two heuristic methods for the minimum sum of completion times objective. Our approach is to group jobs into a batch and then apply a scheduling heuristic to the batches. We show that grouping jobs earlier in the schedule, although intuitive, can result in poor performance when jobs have sufficiently large differences in processing times. We provide bounds on the performance of the algorithms and also present computational results comparing the solutions to the optimal objective obtained from the integer programming formulation for SSP. With a smaller number of jobs, both algorithms produce comparable solutions. For instances with a larger number of jobs and a higher variability in spatial dimensions, we observe that the efficient area model outperforms the iterative model both in terms of solution quality and run time.

Keywords Spatial scheduling • Integer programs • Approximation algorithms • Optimal packings

S. Srinivasan (✉) • J.P. Brooks
Virginia Commonwealth University, Richmond, VA 23284, USA
e-mail: srinivasans3@vcu.edu; jpbrooks@vcu.edu

J.H. Wilson
Northwestern University, Evanston, IL 60208, USA
e-mail: jill.wilson@northwestern.edu

G. Fasano, J.D. Pintér (eds.), *Optimized Packings with Applications*, Springer Optimization and Its Applications 105, DOI 10.1007/978-3-319-18899-7_12

12.1 Introduction

In large-scale production and manufacturing industries, assembly units are often heavy and occupy large areas. Since physical processing space is limited at such facilities, the assembly line scheduling needs to assign non-overlapping locations (spatial characteristic) and starting times (temporal characteristic) for each job. Further, the schedule should ensure that the locations assigned are the same contiguous units of space for the entire duration of processing as jobs cannot be moved once set up. Mathematically, the spatial scheduling problem (SSP) can be described as follows: Given a set J of jobs with processing times p_j, heights h_j, and widths w_j, and a workspace of height H and width W, does there exist a schedule of the jobs that effectively utilizes the workspace such that some scheduling objective is minimized? Figure 12.1 shows the layout of jobs before and after applying spatial scheduling solution procedures. We can see that initially the space is not utilized effectively and some jobs are waiting to be processed. On applying some spatial scheduling method, we get a better utilization of the space and no delays in processing of jobs.

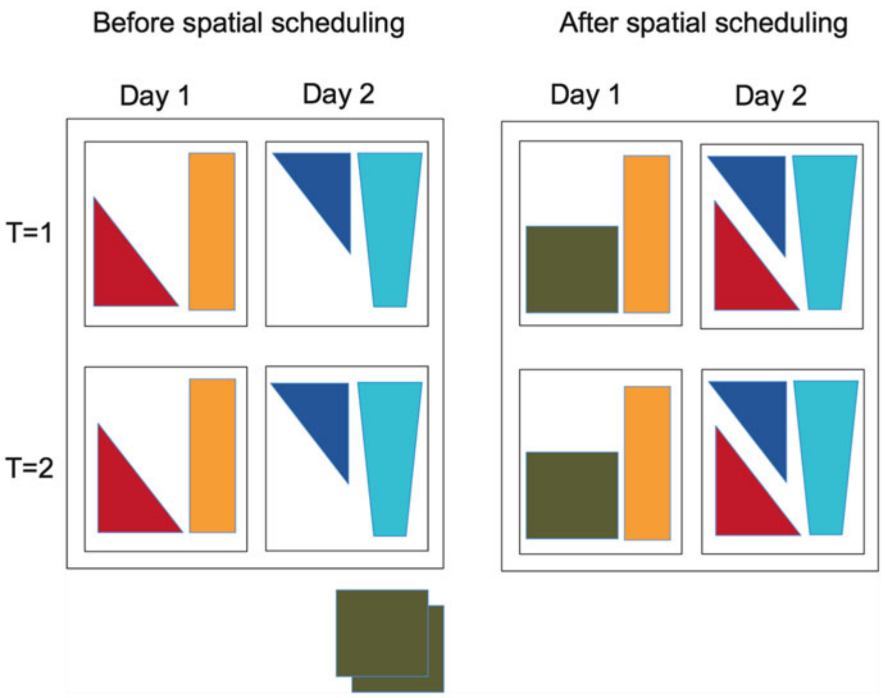

Fig. 12.1 Depicting the motivation for spatial scheduling with jobs each requiring two time units scheduled over a 2-day horizon

When the space required by all jobs are identical to the dimensions of the workspace, the problem reduces to single machine scheduling (SMS) and is polynomially solvable. However, when the dimensions of the jobs are allowed to vary, the spatial constraints add an additional level of complexity to the traditional scheduling problem [5]. Also spatial resources are not divisible and distributable like normal renewable resources. Due to the computational intractability in solving large instances of the problem, there is a need to develop alternate solution methodologies that provide near optimal solutions. Previous work in spatial scheduling has mostly been in the context of shipbuilding applications [4, 11, 15, 18]. Much of the literature focusses on approaches with the objective of minimizing the makespan or $\max_{j \in J} C_j$, where C_j denotes the completion time of job j in a given schedule [2, 9, 16, 22, 23]. Garcia and Rabadi [7] provides a meta heuristic algorithm to minimize the total tardiness for instances with release dates and multiple processing areas. To the best of our knowledge, this is the first study to consider the minimum sum of completion times ($\sum_{j \in J} C_j$) objective for this problem. When jobs are independent and competing for the same resource, the cost associated with individual completion times becomes more relevant and natural [12]. Evaluating completion times for individual jobs also becomes important while measuring the time in the system for each job. The motivation, here, is to examine the most simple form of the problem by considering a workspace area of fixed dimensions. We disallow precedence constraints, due-dates, rotation of jobs, and set-up times. By doing so, we are able to focus on the relationship between the spatial and temporal components in the problem and gain a better understanding of the problem characteristics.

Our approach in developing solution procedures for the problem is to take ideas from two-dimensional bin packing (2DBP) and group jobs similar in processing times to form a batch. Once the batches are determined we can schedule them using some heuristic rule. This approach lets us relax the temporal constraints in the original problem. We identify scenarios where batching can be effective or disadvantageous. For the minimum sum of completion times objective, it seems intuitive to schedule as many jobs as we can ahead in the schedule to produce a lower objective [19]. However, we show that grouping jobs with different processing times earlier in the schedule actually results in an objective value larger than if each job were to be assigned its own batch. We also determine that the sequence in which the batches are scheduled is another factor affecting the objective.

After introducing the problem in Sect. 12.2, we propose two methods (iterative and efficient area) to determine the batches in Sect. 12.3. Section 12.4 analyzes the performance of the batching methods. Computational results comparing the two methods to the optimal objective obtained from the integer programming formulation for SSP are presented in Sect. 12.5.

12.2 The Spatial Scheduling Problem

In this section, we formally introduce the SSP, which involves determining spatial layouts and starting times for a set J of N jobs to be scheduled on a single $W \times H$ work space. Each job $j \in J$ requires a processing time (p_j), width (w_j), and height (h_j). Throughout the reminder of this chapter we consider minimizing the sum of completion times (denoted by Z) as the objective for SSP. We assume that the jobs cannot be rotated or preempted. Also we do not consider due dates, release dates, or precedence relationships for the jobs. Further, without loss of generality, we assume all problem data to be integer. The problem can then be denoted using the following mixed-integer programming (MIP) formulation adapted from [8].

$$\min \sum_{j \in J} z_j \tag{12.1}$$

subject to:

$$-x_i + x_j - W\alpha_{ij} \geq -W + w_i \quad \forall i,j \in J, i \neq j \tag{12.2}$$

$$-y_i + y_j - H\beta_{ij} \geq -H + h_i \quad \forall i,j \in J, i \neq j \tag{12.3}$$

$$-z_i + z_j - T\gamma_{ij} \geq -T + p_i \quad \forall i,j \in J, i \neq j \tag{12.4}$$

$$\alpha_{ij} + \alpha_{ji} + \beta_{ij} + \beta_{ji} + \gamma_{ij} + \gamma_{ji} \geq 1 \quad \forall i,j \in J, i \neq j \tag{12.5}$$

$$-x_i - w_i \geq -W \qquad \forall i \in J \tag{12.6}$$

$$-y_i - h_i \geq -H \qquad \forall i \in J \tag{12.7}$$

$$x_i, y_i, z_i \geq 0 \qquad \forall i \in J \tag{12.8}$$

$$\alpha_{ij}, \beta_{ij}, \gamma_{ij} \in \{0, 1\} \qquad \forall i,j \in J \tag{12.9}$$

where

J is the set of all jobs
x_j is the x-coordinate of job $j \in J$
y_j is the y-coordinate of job $j \in J$
z_j is the z-coordinate (start time) for job $j \in J$

$$\alpha_{ij} = \begin{cases} 1 \text{ if no overlap occurs between jobs i and j in the x direction} \\ 0 \text{ otherwise} \end{cases}$$

$$\beta_{ij} = \begin{cases} 1 \text{ if no overlap occurs between jobs i and j in y direction} \\ 0 \text{ otherwise} \end{cases}$$

$$\gamma_{ij} = \begin{cases} 1 \text{ if no overlap occurs between jobs i and j in z direction} \\ 0 \text{ otherwise} \end{cases}$$

For $i,j \in J$

Here Z is obtained by adding $\sum_{j \in J} p_j$ to the objective in (12.1). Constraints (12.2)–(12.5) prevent overlap from occurring in the x (width), y (height), and z (time) dimensions. We use constraints (12.6) and (12.7) to ensure that the jobs are confined to the physical dimensions of the workspace. Duin and Sluis [5] shows that scheduling problems with varying spatial resource requirements are NP Hard. Hence obtaining optimal solutions to large problem instances is computationally intractable. Therefore, the motivation here is to develop methods that provide provably good solutions to minimize Z for large instances of SSP, quickly and efficiently. Approximation algorithms deliver solutions with provable quality that are bounded in runtime. The following definition of an approximation algorithm can be found in [20, 21]. Suppose we wish to solve an NP-hard minimization problem consisting of instances in $\mathscr{I}$. Let $z(I)$=min$\{c_I x \ : \ x \in S_I\} \ \forall I \in \mathscr{I}$. Let $\mathscr{A}$ be an algorithm that operates on instances in $\mathscr{I}$, and let $\mathscr{A}(I)$ be the objective value resulting from the application of $\mathscr{A}$ to I. Let $\rho \geq 1$.

Definition 1. $\mathscr{A}$ is a ρ-approximation algorithm for $\mathscr{I}$ if for each $I \in \mathscr{I}$, $\mathscr{A}$ runs in time polynomial in the size of I, and $\mathscr{A}(I) \leq \rho z(I)$. $\mathscr{A}$ is said to have a factor ρ, also referred to as the performance guarantee of $\mathscr{A}$.

Observe that we compare the objective value obtained by the application of the algorithm to instance I with the optimal objective value $z(I)$ for that instance. In practice, however, this is not possible, because if $z(I)$ is known then there would be no need to approximate it. To overcome this issue and calculate ρ, we compare $\mathscr{A}(I)$ with a lower bound for $z(I)$, say $L(I)$. Lower bounds can be obtained using LP or combinatorial relaxations. Since $L(I) \leq z(I)$ we have

$\mathscr{A}(I) \leq \rho L(I) \implies \mathscr{A}(I) \leq \rho z(I)$.

In the following sections, we describe the development of an approximation algorithm (with two variants) based on existing packing algorithms and discuss its performance.

12.3 Batch-Scheduling

12.3.1 Introduction

SSP requires that jobs be arranged without overlap in a two-dimensional space while minimizing some scheduling objective. The spatial component of SSP can be attributed to optimized multi-dimensional packing problems. Lodi et al. [13] provides a survey of the models and algorithms used to solve the 2DBP problem. Castillo et al. [3] presents applications and approaches to solve circle packing problems encountered in container loading. Batch-scheduling ideas originated from the problem of scheduling "burn-in" operations at large-scale integrated circuit manufacturing [1, 10]. Mathirajan and Sivakumar [14] surveys the literature for scheduling of batching processors in the semi-conductor industry. The central idea in batch-scheduling is grouping similar jobs together to form a "batch." All jobs in a batch start at the same time and the next batch starts upon completion of the longest job in the previous batch. The processing time of a batch is equal to the largest processing time of any job in the batch. Our goal is to utilize ideas from 2DBP to design batch-scheduling strategies that identify the batches consisting of jobs that can simultaneously fit the space to minimize the sum of completion times.This approach lets us to relax the temporal constraints in the original problem.

Assume we have a set J of N jobs such that $p_1 \leq p_2 \leq \cdots \leq p_N$. When all the jobs fit in the space simultaneously, irrespective of the difference in their processing times p_j they are placed in the same batch. So $Z = \sum_{j \in J} p_j$. If no pair of jobs simultaneously fits the space, SSP reduces to SMS. Then each job is its own batch and $Z = \sum_{j=1}^{N} \sum_{i=1}^{j} p_i$. Smith [19] proved that ordering jobs in the nondecreasing sequence of their processing times is optimal for SMS. In general, while minimizing the sum of completion times, the more jobs we can fit earlier in our schedule the lower the objective. Therefore, it seems intuitive to always group jobs together rather than assign them to individual batches. Consider an instance of SSP with W=H=3 and job data as given in Table 12.1. Jobs 1 and 3 are the only jobs that fit the space simultaneously.

Let the processing times $[p_1, p_2, p_3]$ = [2, 3, 7] and let us assume we schedule the batch with the lowest processing time first. We define batch processing time as the maximum processing time of jobs in a batch. Therefore, the batch sequence is $\{2\}$ and $\{1, 3\}$ as seen in Fig. 12.2a. Then the objective value for batched jobs is calculated as $Z = p_1 + 3p_2 + p_3$ =2 + 9 + 7 = 18. Alternately, if we schedule the batches in their own batch, the sequence is $\{1\}, \{2\}, \{3\}$ as seen in Fig. 12.2b and Z = $3p_1 + 2p_2 + p_3$ = 6 + 6 + 7 = 19. This shows that grouping jobs can result in a lower sum of completion times objective.

Now suppose, $[p_1, p_2, p_3]$ = [2, 24, 25]. When jobs 1 and 3 are batched, Z = $p_1 + 3p_2 + p_3$ = 2 + 72 + 25 = 99. Without batching, Z = $3p_1 + 2p_2 + p_3$ = 6 + 49 + 25 = 79. Thus in scenarios where jobs with large differences in processing times are grouped together, the batching approach does not necessarily lead to improvement in the objective.

Table 12.1 Example instance with three jobs such that only jobs 1 and 3 simultaneously fit the space

Job	Width	Height
1	3	2
2	2	3
3	3	1

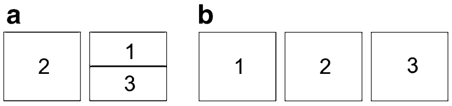

Fig. 12.2 Batching sequence for example with three jobs. (**a**) Batching. (**b**) No batching

Proposition 1. *For N jobs, assume that $p_1 < p_2 < \cdots < p_{N-1} < p_N$. When jobs with both the largest and smallest processing times are assigned to the same batch, that is $\{1, N\}$ form a batch, and $(N-1)p_1 - p_2 - \cdots - p_{N-1} < 0$, the sum of completion times obtained by batching is greater than the objective value obtained without batching.*

Proof. Let $\sum_{j \in J} C_j^b$ be the sum of completion times obtained when batching and $\sum_{j \in J} C_j^n$ be the sum of completion times obtained without batching. Jobs $\{1, N\}$ form a batch, while the other jobs are each assigned individual batches. Since $p_1 < p_2 < \cdots < p_{N-1} < p_N$, batch $\{1, N\}$ is processed at the end of the schedule (see Fig. 12.3). So $\sum_{j \in J} C_j^b = p_1 + Np_2 + \cdots + 3p_{N-1} + p_N$. If each job is assigned its own batch, then $\sum_{j \in J} C_j^n = Np_1 + (N-1)p_2 + \cdots + 2p_{N-1} + p_N$. Therefore,

$$\begin{aligned}
&\sum_{j \in J} C_j^b - \sum_{j \in J} C_j^n \\
&= [p_1 + Np_2 + \cdots + 3p_{N-1} + p_N] - [Np_1 + (N-1)p_2 + \cdots + 2p_{N-1} + p_N] \\
&= (N-1)p_1 - p_2 - \cdots - p_{N-1}
\end{aligned}$$

Hence, when $(N-1)p_1 - p_2 - \cdots - p_{N-1} < 0$, the result follows.

This contradicts the notion of scheduling as many jobs earlier in the schedule to minimize our objective. So, our intuitions about general scheduling problems do not always apply directly to problems with spatial resources. Batching seems to be beneficial only when processing times are similar.

When looking at the instance with $[p_1, p_2, p_3] = [2, 24, 25]$, we observed that scheduling batches in the sequence $\{2\}$ then $\{1, 3\}$ as seen in Fig. 12.4a results in an objective $Z = p_1 + 3p_2 + p_3 = 2 + 72 + 25 = 99$. Instead, if we were to schedule the batches in the sequence $\{1, 3\}$ then $\{2\}$, as seen in Fig. 12.4b, the objective value is calculated as $Z = p_1 + p_2 + 2p_3 = 2 + 24 + 50 = 76$. This suggests that scheduling the jobs in the increasing order of batch processing times is not always effective.

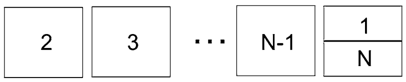

Fig. 12.3 Sequence in which batches are scheduled for Proposition 1

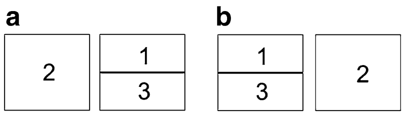

Fig. 12.4 Comparing strategies for sequencing batches. (**a**) Batch sequence. (**b**) Alt. sequence

Proposition 2. *Consider a set J of N jobs such that $p_1 < p_2 < \cdots < p_{N-1} < p_N$. If m of those jobs are in a batch B, including Job 1 and Job N, and $p_N > mp_i, \forall i \in J \setminus B$, then placing batch B at the end of a schedule provides a better objective than placing it at the beginning of the schedule.*

Proof. Let $\sum\limits_{j \in J} C_j^b$ be the sum of completion times obtained when processing batch B at the end of the schedule and $\sum\limits_{j \in J} C_j^a$ be the sum of completion times obtained using an alternate sequencing of batches (batch B is the first batch to be scheduled). Jobs $\{1, N\}$ along with $(m-2)$ other jobs form a batch B, while the remaining jobs are each assigned individual batches. Let $\{1, u_1, u_2, \cdots, u_{m-2}, N\}$ be the m jobs in batch B such that

$p_1 < p_2 < \cdots < p_{u_1-1} < p_{u_1} < \cdots < p_{u_{m-2}} < p_{u_{m-1}} < \cdots < p_N$.

The sequence of batches scheduled in increasing order of batch processing times is $\{2\}, \{3\}, \cdots, \{u_1 - 1\}, \cdots, \{u_{m-1}\}, \cdots, \{N-1\}, \{B\}$.

So $\sum\limits_{j \in J} C_j^b = (p_1 + p_{u_1} + \cdots + p_{u_{m-2}} + p_N) + (Np_2 + \cdots + (m+1)p_{N-1})$.

Alternately, if we place batch B at the beginning of the schedule, $\sum\limits_{j \in J} C_j^a$ is given by

$p_1 + p_{u_1} + \cdots + p_{u_{m-2}} + (N-m+1)p_N + \cdots + p_{N-1}$.

Therefore, $\sum\limits_{j \in J} C_j^a - \sum\limits_{j \in J} C_j^b$

$$= \begin{aligned}&[(p_1 + p_{u_1} + \cdots + p_{u_{m-2}} + (N-m+1)p_N + (N-m)p_2 + \cdots + p_{N-1})] - \\ &\qquad [(p_1 + p_{u_1} + \cdots + p_{u_{m-2}} + p_N) + (Np_2 + \cdots + (m+1)p_{N-1})]\end{aligned}$$

$$= -mp_2 - mp_3 - \cdots - mp_{N-1} + (N-m)p_N$$

Hence, when $p_N > mp_2, p_N > mp_3, \cdots, p_N > mp_{N-1}$, the result follows.

In summary, placing jobs with the smallest and largest processing times in the same batch or scheduling jobs in the increasing order of batch processing times does not necessarily result in a good batching scheme.

12.3.2 Forming the Batches

Using the insights gained from our previous analysis, we group jobs similar in processing time that also efficiently utilize the space to form a batch. We present two MIP models, iterative and efficient area, that identify the assignment of jobs to batches. The objective for the iterative model is to minimize the maximum difference in processing times among jobs for each batch. The efficient area model extends this idea by also minimizing the total unused area in each batch. Both MIP formulations have been adapted from the 2DBP model found in [17]. Let J denote the set of jobs and B the set of batches. Since at most each job can be its own batch, the number of batches equals the number of jobs (N).

12.3.2.1 Iterative Model

In the iterative model (M1), we add a constraint to limit the number of batches (S) being used by the model. We do not chose S as part of the model, because the objective here is not to reduce the number of batches used, but to find the best assignment of jobs (to batches) that minimizes the sum of completion times. Therefore, the strategy is to iterate through possible values for S, starting at $S = N - 1$ and decreasing by 1 in each iteration. From the set of all solutions, we can then chose the batching that results in the lowest sum of completion times objective value. The formulation for the iterative model is given by

$$\min \sum_{b \in B} (Zmax_b - Zmin_b) \tag{12.10}$$

$$\sum_{b \in B} r_{jb} = 1 \qquad \forall j \in J \tag{12.11}$$

$$x_j + w_j \leq W \qquad \forall j \in J \tag{12.12}$$

$$y_j + h_j \leq H \qquad \forall j \in J \tag{12.13}$$

$$x_i + w_i - x_j \leq W(1 - l_{ij}) \quad \forall i,j \in J, i < j, b \in B \tag{12.14}$$

$$y_i + h_i - y_j \leq H(1 - b_{ij}) \quad \forall i,j \in J, i < j, b \in B \tag{12.15}$$

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} + (1 - r_{ib}) + (1 - r_{jb}) \geq 1 \qquad \forall i,j \in J, b \in B \tag{12.16}$$

$$Zmin_b \leq (p_j - M)r_{jb} + Mq_b \qquad \forall j \in J, b \in B \tag{12.17}$$

$$Zmax_b \geq p_j r_{jb} \qquad \forall j \in J, b \in B \tag{12.18}$$

$$r_{jb} \leq q_b \qquad \forall j \in J, b \in B \tag{12.19}$$

$$\sum_{j \in J} r_{jb} - \epsilon q_b \geq 0 \qquad \forall b \in B \tag{12.20}$$

$$\sum_{b \in B} q_b = S \tag{12.21}$$

$$x_j, y_j \geq 0 \qquad \forall j \in J \tag{12.22}$$

$$Zmin_b, Zmax_b \geq 0 \qquad \forall b \in B \tag{12.23}$$

$$l_{ij}, b_{ij} \in \{0, 1\} \qquad \forall i,j \in J \tag{12.24}$$

$$r_{jb} \in \{0, 1\} \qquad \forall j \in J, b \in B \tag{12.25}$$

$$q_b \in \{0, 1\} \qquad \forall b \in B \tag{12.26}$$

where

J is the set of all jobs

B is the set of all batches

x_j is the x-coordinate of job $j \in J$
y_j is the y-coordinate of job $j \in J$
$Zmax_b$ is the maximum processing time of jobs in batch $b \in B$
$Zmin_b$ is the minimum processing time of jobs in batch $b \in B$

$$r_{jb} = \begin{cases} 1 \text{ if job j is in batch b} \\ 0 \text{ otherwise} \end{cases}$$

$$l_{ij} = \begin{cases} 1 \text{ if job i is to the left of job j} \\ 0 \text{ otherwise} \end{cases}$$

$$b_{ij} = \begin{cases} 1 \text{ if job i is below job j} \\ 0 \text{ otherwise} \end{cases}$$

$$q_b = \begin{cases} 1 \text{ if batch b is nonempty} \\ 0 \text{ otherwise} \end{cases}$$

For $i, j \in J$ and $b \in B$.

Here, constraint (12.11) ensures that each job is assigned to only one batch. Constraints (12.12) and (12.13) ensure that jobs do not exceed the width and height of the space. We use constraints (12.14)–(12.16) to prevent overlap of jobs within the space. Constraint (12.17) determines the minimum processing time within a batch, while (12.18) identifies the maximum processing time for each batch. If job j is in batch b ($r_{jb} = 1$), then constraint (12.19) makes sure batch b is non-empty ($q_b = 1$). When no jobs are present in a batch, constraint (12.20) ensures that the batch is empty or $q_b = 0$. Constraint (12.21) sets the number of batches to be used by the model to some value S. We set $\epsilon = 0.5$ and define $M = 1 + \max_{j \in J} p_j$.

12.3.2.2 Efficient Area Model

While solving $N - 1$ instances of M1 for different values of S finds the best possible batch assignment, the second approach or efficient area model (M2) proposes to solve just one MIP to decide when and where to place jobs. The efficient area model includes an area utilization component to the existing objective. So, model 2 minimizes the maximum difference in processing times and the amount of workspace area that remains unused for each batch. The formulation for the efficient area model is given by

$$\min \sum_{b \in B} (Zmax_b - Zmin_b + UA_b) \tag{12.27}$$

$$\sum_{b \in B} r_{jb} = 1 \qquad \forall j \in J \tag{12.28}$$

$$x_j + w_j \leq W \qquad \forall j \in J \tag{12.29}$$

$$y_j + h_j \leq H \qquad \forall j \in J \tag{12.30}$$

$$x_i + w_i - x_j \leq W(1 - l_{ij}) \quad \forall i, j \in J, i < j, b \in B \tag{12.31}$$

$$y_i + h_i - y_j \leq H(1 - b_{ij}) \quad \forall i, j \in J, i < j, b \in B \tag{12.32}$$

$$l_{ij} + l_{ji} + b_{ij} + b_{ji} + (1 - r_{ib}) + (1 - r_{jb}) \geq 1 \quad \forall i, j \in J, b \in B \tag{12.33}$$

$$Zmin_b \leq (p_j - M) r_{jb} + M q_b \quad \forall j \in J, b \in B \tag{12.34}$$

$$Zmax_b \geq p_j r_{jb} \quad \forall j \in J, b \in B \tag{12.35}$$

$$r_{jb} \leq q_b \quad \forall j \in J, b \in B \tag{12.36}$$

$$\sum_{j \in J} r_{jb} - \epsilon q_b \geq 0 \quad \forall b \in B \tag{12.37}$$

$$WHq_b - \sum_{j \in J} w_j h_j r_{jb} = UA_b \quad \forall b \in B \tag{12.38}$$

$$x_j, y_j \geq 0 \quad \forall j \in J \tag{12.39}$$

$$Zmin_b, Zmax_b, UA_b \geq 0 \quad \forall b \in B \tag{12.40}$$

$$l_{ij}, b_{ij} \in \{0, 1\} \quad \forall i, j \in J \tag{12.41}$$

$$r_{jb} \in \{0, 1\} \quad \forall j \in J, b \in B \tag{12.42}$$

$$q_b \in \{0, 1\} \quad \forall b \in B \tag{12.43}$$

where
J is the set of all jobs
B is the set of all batches
x_j is the x-coordinate of job $j \in J$
y_j is the y-coordinate of job $j \in J$
$Zmax_b$ is the maximum processing time of jobs in batch $b \in B$
$Zmin_b$ is the minimum processing time of jobs in batch $b \in B$
UA_b is the unused area in batch $b \in B$

$$r_{jb} = \begin{cases} 1 \text{ if job j is in batch b} \\ 0 \text{ otherwise} \end{cases}$$

$$l_{ij} = \begin{cases} 1 \text{ if job i is to the left of job j} \\ 0 \text{ otherwise} \end{cases}$$

$$b_{ij} = \begin{cases} 1 \text{ if job i is below job j} \\ 0 \text{ otherwise} \end{cases}$$

$$q_b = \begin{cases} 1 \text{ if batch b is nonempty} \\ 0 \text{ otherwise} \end{cases}$$

For $i, j \in J$ and $b \in B$.

Here, constraint (12.28) ensures that each job is assigned to only one batch. Constraints (12.29) and (12.30) ensure that jobs do not exceed the width and height of the space. We use constraints (12.31)–(12.33) to prevent overlap of jobs within the space. Constraint (12.34) determines the minimum processing time within a batch, while (12.35) identifies the maximum processing time for each batch. If job

j is in batch b ($r_{jb} = 1$), then constraint (12.36) makes sure batch b is non-empty ($q_b = 1$). When no jobs are present in a batch, constraint (12.37) ensures that the batch is empty or $q_b = 0$. Constraint (12.38) calculates the unused area for each batch b. We set $\epsilon = 0.5$ and define $M = 1 + \max_{j \in J} p_j$.

12.3.3 Scheduling the Batches

Once the batches are identified using either M1 or M2, it is also important to decide the sequence in which to schedule the batches. Smith [19] proved that the shortest processing time (SPT) rule, ordering jobs in the nondecreasing sequence of their job processing times, is optimal for the SMS problem. The idea is that by scheduling shorter jobs earlier in the schedule, more jobs can finish early resulting in a smaller sum. For SSP, the rule translates to scheduling the batches in the nondecreasing sequence of their batch processing times. For example, if P_1 is the maximum processing time of all jobs in batch 1 and P_2 is the maximum processing time of all jobs in batch 2, then batch 1 is scheduled before batch 2 if and only if $P_1 \leq P_2$. However, as noted before, there are instances for which this rule does not necessarily provide a better objective value. Therefore, we also consider scheduling jobs in the non-decreasing order of the average batch processing times, or the average processing time of all the jobs in a batch. We indicate the two scheduling rules as MAX and AVG, respectively.

12.3.4 Post Processing Algorithm

By solving each instance of SSP using the iterative and efficient area models, we determine the assignments of jobs to batches that minimize the maximum difference in processing times while efficiently utilizing the workspace. With this information, we then schedule the batches by applying either the MAX or AVG rules. Once a schedule is created, we calculate the sum of completion times for the jobs as $Z_H = \sum_{j \in J} C_j^H$, where C_j^H is the completion time for job j. With this batching algorithm, each job must wait until the previous batch has completed before it can start processing. In reality there may be jobs in the current batch that finish processing before the final job in the batch. This means that jobs in later batches may be able to start earlier in the schedule. Since neither MIP model takes into account the temporal dimension, we use a post-processing algorithm to incorporate this observation and improve Z_H. For each batch the algorithm determines if jobs can start processing earlier in the schedule. If job j can be moved ahead in time by say t_j units, then the completion time is updated as, $\hat{C}_j = C_j^H - t_j$ and $\hat{Z} = \sum_{j \in J} \hat{C}_j$ is the new objective value.

Proposition 3. *For instances defined by N=nk jobs, $n, k \in \mathbb{Z}_+^*$, where $w_j = \frac{W}{k}$, $h_j = H \; \forall \, j \in J$, and $p_1 \le p_2 \le \cdots \le p_N$, the solution obtained after the post-processing routine is optimal.*

Proof. Consider the instances with $N = nk$ jobs, such that k jobs can simultaneously fit the space. Let C_j^H, $\hat{C}_j$, and C_j^{OPT} denote the completion time for job j and Z_H, $\hat{Z}$, and Z_{OPT} denote the objective value for the batch-scheduling algorithm, the post-processing routine, and the optimal solution, respectively. First we observe that if k jobs can simultaneously fit within the workspace that there are n batches. So for all jobs $j \le k$, $\hat{C}_j = C_j^{OPT}$.

Let U = $\{u_1, u_2, \cdots, u_k\}$ denote the k jobs in the next batch waiting to be scheduled, such that $p_{u_1} \le p_{u_2} \le \cdots \le p_{u_k}$. Then by definition, if job j can be moved ahead in time by say t_j units, the new completion time is given by, $\hat{C}_j = C_j^H - t_j$. Since job u_i can be processed as soon as u_{i-k} completes and space becomes available, we get the following recursive improvement on job completion times:

$\hat{C}_{u_i} = C_{u_i}^H - [(p_{u_i-1} - p_{u_i-k}) + \cdots + (p_k - p_1)] \; \forall i \in \{1, \cdots, k-1\}$ and

$\hat{C}_{u_k} = C_{u_k}^H$

So, $\hat{Z} = Z_H - \sum_{j=1}^{n} \sum_{i=1}^{j} (p_{ik} - p_{(ik-k+1)}) = Z_{OPT}$

12.4 Performance Analysis

In this section, we present solution guarantees on the objective values Z_H generated by both the batch-scheduling algorithms. We refer to Z_{OPT} as the optimal objective for the SSP formulation. We begin by analyzing special instances of SSP with a set J of N jobs such that at any given time k jobs can simultaneously fit the space $(W \times H)$ and $p_1 \le p_2 \le \cdots \le p_N$.

Theorem 1. *Suppose there are N=nk jobs for any $n, k \in \mathbb{Z}_+^*$, $w_j \le W$ and $h_j \le H$ $\forall \, j \in J$, and $p_1 \le p_2 \le \cdots \le p_N$, where k jobs can simultaneously fit the space, then batch-scheduling is a k-approximation algorithm.*

Proof. Let J denote the set of nk jobs and B the set of batches. If the first k jobs are scheduled in a batch at the beginning of the schedule, job $k+1$ does not start until any of the jobs finish processing. The first job to finish processing would be job 1. So completion time, $C_{k+1} = p_{k+1} + p_1$. Applying this reasoning we note that a lower bound on the optimal objective for these instances is given by, $Z_{OPT} \ge \sum_{j=1}^{n} (n - j + 1)(p_{jk} + p_{jk-1} + \cdots + p_{jk-k+1})$, since $p_1 \le p_2 \le \cdots \le p_N$ and we are trying to minimize the sum of completion times. In the following discussion p_{jk} is defined as the processing time of the job in the j times k position in the sequence $p_1 \le p_2 \le \cdots \le p_N$.

Since only k jobs can occupy the space at any given time, the number of batches is $\frac{nk}{k} = n$. If we use the MAX rule, $Z_b = max_{j \in b} p_j$ for each batch $b \in B$ and $Z_1 \le Z_2 \le \cdots \le Z_n$. Let us order the jobs in the sequence of the batches they are assigned and in the increasing order of their processing times within each batch, so that $p_{|jk|}$ refers to the processing time of the jkth job in the scheduling sequence and not p_{jk}. The completion time of job j, C_j^H, based on this new ordering is then calculated as the sum of its processing time and the completion times of the batches scheduled ahead of it. For example, if jobs j is in batch b, the completion time is calculated as: $C_j^H = p_j + Z_{b-1} + \cdots + Z_1$.

$$
\begin{aligned}
Z_H &= \sum_{j \in J} C_j^H && (12.44)\\
&= \sum_{j \in J} p_j + kZ_1 + k(Z_1 + Z_2) + k(Z_1 + Z_2 + Z_3) + \cdots + \\
&\quad + k(Z_1 + Z_2 + \ldots + Z_{n-1}) && (12.45)\\
&= \sum_{j \in J} p_j + k[(n-1)Z_1 + (n-2)Z_2 + \cdots + 2Z_{n-2} + Z_{n-1}] && (12.46)\\
&= \sum_{j \in J} p_j + k[(n-1)p_{|k|} + (n-2)p_{|2k|} + \cdots + \\
&\quad + 2p_{|(n-2)k|} + p_{|(n-1)k|}] && (12.47)\\
&= \sum_{j=1}^{n} (p_{|jk-1|} + \cdots + p_{|jk-k+1|}) + \sum_{j=1}^{n} ((nk - jk + 1)p_{|jk|}) && (12.48)\\
&= \sum_{j=1}^{n} (p_{|jk-1|} + \cdots + p_{|jk-k+1|}) + k\sum_{j=1}^{n} ((n - j + \frac{1}{k})p_{|jk|}) && (12.49)\\
&= \sum_{j=1}^{n} (p_{|jk-1|} + \cdots + p_{|jk-k+1|}) + \sum_{j=1}^{n} ((n - j + \frac{1}{k})p_{|jk|}) \\
&\quad + (k-1)\sum_{j=1}^{n} ((n - j + \frac{1}{k})p_{|jk|}) && (12.50)\\
&\le \sum_{j=1}^{n} (p_{|jk-1|} + \cdots + p_{|jk-k+1|}) + \sum_{j=1}^{n} ((n - j + \frac{1}{k})p_{|jk|}) \\
&\quad + (k-1)\sum_{j=1}^{n} ((n - j + 1)p_{|jk|}) && (12.51)\\
&\le \sum_{j=1}^{n} (p_{|jk-1|} + \cdots + p_{|jk-k+1|}) + \sum_{j=1}^{n} ((n - j + \frac{1}{k})p_{|jk|}) \\
&\quad + (k-1)Z_{OPT} && (12.52)
\end{aligned}
$$

$$\leq Z_{OPT} + (k-1)Z_{OPT} \tag{12.53}$$

$$= kZ_{OPT} \tag{12.54}$$

Equation (12.45) is obtained from the definition of completion times, C_j^H and we get Eq. (12.46) per the definition of Z_b. In each batch b of k jobs, the batch processing time is the processing time of the kth job in the batch, $p_{|bk|}$. This is the only processing time included in the calculation of completion times for the batches scheduled later. The processing times of the remaining $(k-1)$ jobs are not repeated in this objective as seen in Eq. (12.47). Equations (12.52) and (12.53) follow from the lower bound on the optimal objective, $Z_{OPT} \geq \sum_{j=1}^{n}(n-j+1)(p_{jk}+p_{jk-1}+\cdots+p_{jk-k+1})$.

The bound shown helps us understand what makes instances of SSP hard. The real difficulty in solving instances of SSP lies in the spatial constraints as reflected by the bound, which is dependent on k, the number of jobs that can simultaneously fit within the given workspace. Also, recall that when minimizing the sum of completion times, we want to schedule more jobs earlier in the schedule. This is because the completion time of a job includes the completion times of the jobs earlier in the schedule. When $k = 1$, SSP reduces to SMS and our batching heuristic becomes SPT, which we know is optimal [19]. Our bound depicts that as k increases, the spatial component plays a larger role in the objective obtained from the batch-scheduling algorithm.

Consider the instance data with six jobs shown in Table 12.2 and a 10×10 workspace. We can fit three (k) jobs within the space, so the batches formed are $\{1,2,3\}$ and $\{4,5,6\}$ as shown in Fig. 12.5a. The sum of completion times before post-processing, $Z_H = p_1 + p_2 + 4p_3 + p_4 + p_5 + p_6 = 192$. Using the lower bound we know that $Z_{OPT} \geq 2(p_1+p_2+p_3) + (p_4+p_5+p_6) = 153$. So, $Z_H \leq 3Z_{OPT}$.

Now, if we were to schedule the batches as seen in Fig. 12.5b in the sequence $\{1,2\}$, $\{3,4\}$, and $\{5,6\}$ such that k=2, then $Z_H = p_1 + 5p_2 + p_3 + 3p_4 + p_5 + p_6 = 181$. Therefore, packing more jobs (larger k) that are sufficiently different in processing times because they efficiently utilize the space does not result in a lower sum of completion times objective.

Table 12.2 SSP instance with N = 6 jobs to depict that grouping more jobs in a batch does not guarantee lower objective value

Job	Processing time	Width	Height
1	1	2	H
2	2	4	H
3	21	2	H
4	22	4	H
5	41	2	H
6	42	4	H

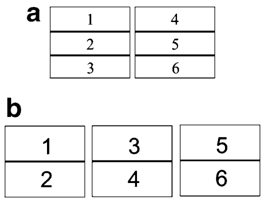

Fig. 12.5 Example schedule with two and three jobs in a batch. (**a**) three job batch (**b**) two job batch

12.5 Computational Analysis

In this section we provide the computational results obtained by evaluating the two proposed procedures for solving the SSP and comparing it to the optimal solution or the best solution obtained after a certain time limit for the integer programming formulation of SSP.

12.5.1 Instance Generation

We tested both the iterative model (M1) and the efficient area model (M2) on generated instances of SSP. The instance class denoted as $NnPpRr < ABC > i$ has n= 5 or 10 jobs, processing times generated in the discrete uniform interval of $(1, p)$ with workspace area dimension $W = H = r$. The value for r is 10 or 20 units and i is an instance indicator. A, B, C classifiers are used to indicate the distributions from which the width and height of jobs are sampled.

Class A $w_j \in$ Uniform Discrete $[1,\frac{W}{2}]$ and $h_j \in$ Uniform Discrete $[1, \frac{H}{2}]$
Class B $w_j \in$ Uniform Discrete $[1,\frac{W}{2}]$ and $h_j \in$ Uniform Discrete $[\frac{H}{2}$,H]
Class C $w_j \in$ Uniform Discrete $[\frac{W}{2}$,W] and $h_j \in$ Uniform Discrete $[\frac{H}{2}$,H]

Five instances of each class-type were generated, resulting in a total of 60 instances. All of the instances had jobs sorted in the increasing order of processing times. Instances in Class C have jobs that occupy more than half the area. This results in each job getting its individual batch and SSP reduces to SMS which can be solved to optimality. So for the computational analysis we only consider instances in classes A and B. By design, instances in Class B should be relatively harder to solve than instances in class A. This is because all of the jobs in class A are small compared to the dimensions of the workspace, so we can fit more jobs together. Difficult instances of the problem occur, when some jobs are small and some are large (Class B).

Larger instances were modified from [6]. The instances have 100, 500, and 1,000 jobs with a 10×7 workspace. For each job:

$w_j \in \mathit{UniformDiscrete}[1, 10]$
$h_j \in \mathit{UniformDiscrete}[1, 7]$
$p_j \in \mathit{UniformDiscrete}[5, 25]$

Since we did not permit rotation of jobs, we had to interchange the widths and heights in certain cases to ensure that the jobs would fit within the space.

12.5.2 Initial Feasible Solution Heuristic

The motivation behind creating the batching models (M1 and M2) was to reduce the complexity of the original SSP by looking only at the packing component of the problem. Nevertheless, we need to understand that M1 and M2 are still MIPs and as the instances grow larger, these models could take longer to solve to optimality. Further, an optimal solution to the batching model does not necessarily guarantee an optimal solution to SSP. In order to improve the solution time for these MIP formulations, we provide the solver with an initial feasible solution obtained from a greedy packing heuristic. Basically, we start with an instance of SSP sorted in the increasing order of job processing times, i.e. $p_1 \leq p_2 \leq \cdots \leq p_N$. We sequentially begin grouping jobs into a batch until they fit the space. Once the job can no longer fit the space, we create a new batch. This process is repeated until all jobs are assigned a batch.

12.5.3 Computational Results

In this section, we compare the solutions generated by the batch-scheduling approaches (iterative and efficient area models) to the optimal solution (OPT) obtained by solving the mixed-integer program for SSP. The batching MIPs, M1 and M2, and the SSP MIP formulation were all implemented using the C programming language and solved using Gurobi 5.0 with a thread count of 1 and cuts parameter set to default on a RedHat Enterprise 6.5 x86_64 server. The following tables compare the objective values and runtimes for the small instances with 5 jobs or 10 jobs and the large instances with 25 jobs or 100 jobs (defined at the beginning of Sect. 12.5).

Table 12.3 lists the objective values obtained from solving instances with five and ten jobs for M1 and M2 using the MAX rule and the optimal solution (OPT) for the original MIP formulation of SSP. Note that the objective reported for M1 is the best possible value among the $N - 1$ potential solutions it obtains and the run time is the total time taken to iteratively solve all of the models. We observe that M2 seems to perform at least as well as M1, and both models return values close to the optimal

Table 12.3 Comparison of objectives obtained from M1, M2, and OPT for small instances of batch-scheduling

Instance	M1 (Best)	M2	OPT	Factors	
				M1/OPT	M2/OPT
N5P10R10A	27	27	27	1.00	1.00
N5P19R10B	33	33	29	1.14	1.14
N5P10R20A	26	26	26	1.00	1.00
N5P10R20B	26	25	23	1.13	1.12
N10P10R10A	49	49	49	1.00	1.00
N10P10R10B	80	72	66	1.22	1.10
N10P10R20A	54	54	51	1.05	1.05
N10P10R20B	101	88	77	1.31	1.14

Table 12.4 Comparison of M1, M2, and OPT runtimes for small instances of batch-scheduling

Instance	Runtime (s)		
	M1 (Total)	M2	OPT
N5P10R10A	0.19	0.01	0.01
N5P19R10B	0.14	0.04	0.02
N5P10R20A	0.17	0.01	0.01
N5P10R20B	0.13	0.07	0.01
N10P10R10A	43.98	0.11	0.03
N10P10R10B	110.21	82.79	287.69
N10P10R20A	300.81	0.23	0.30
N10P10R20B	223.26	114.17	244.33

solution. For these set of instances, the objective values returned by both models for the MAX and AVG rules were identical for instances with five jobs and ten jobs.

Table 12.4 presents the runtimes for solving the instances with five and ten jobs using M1, M2, and the original MIP formulation. We observe that with smaller number of jobs, all three methods produce results quickly. The runtimes for M1 are larger because it iteratively solves $N - 1$ models for each instance with N jobs.

Table 12.5 lists the objective values obtained from solving larger instances (25 and 100 jobs) for M1 and M2 using the MAX rule and the objective Z_{IP} for the original MIP formulation of SSP. Note that the objective reported for M1 is the best possible value among the $N - 1$ potential solutions it obtains, with each iteration of M1 allowed 2 min of execution time. M2 and Z_{IP} report the best objective obtained after 20 min of execution. To improve upon the solution, M2 is given an initial feasible solution. The resulting solution is then updated using the post-processing algorithm. Although both models produce objectives close to optimal, it is observed that with a larger number of jobs, M2 outperforms M1, and on some occasions, after 20 min, M2 is able to produce better solutions than the original SSP formulation.

Table 12.5 Comparison of objectives obtained from M1, M2, and Z_{IP} for large instances of batch-scheduling

	Objective			Factor	
Instance	M1 (Best)	M2 (Updated)	Z_{IP}	M1/Z_{IP}	M2/Z_{IP}
N25P25E11	1,697	1,421	1,215	1.40	1.17
N25P25E12	1,518	1,409	1,022	1.49	1.38
N25P25E13	2,204	2,046	1,540	1.43	1.33
N25P25E14	1,555	1,292	995	1.56	1.30
N25P25H11	1,819	1,762	1,353	1.34	1.30
N25P25H12	1,587	1,332	965	1.64	1.38
N25P25H13	1,929	1,712	1,169	1.65	1.46
N25P25H14	1,625	1,525	1,066	1.52	1.43
N100P25E1	34,372	24,495	28,205	1.22	0.87
N100P25H1	45,571	24,919	27,672	1.65	0.90

In conclusion, the efficient area model seems to be more effective for larger instances both in terms of runtime and solution quality. Further investigations on the weights in the multi-objective function in the efficient area model (M2) could result in potential improvements in objective value.

12.6 Conclusions

The study aims to develop solution methods for SSP with good approximations for the minimum sum of completion times objective. We conclude by summarizing the main contributions and key results presented and by suggesting possible directions for future research. We explored the relationship between the spatial and temporal components of the problem. We considered just the spatial restrictions and utilized bin-packing strategies to identify batches of jobs that will efficiently utilize the space. We then scheduled the jobs using rules to minimize the sum of completion times objective.

When minimizing the sum of completion times objective sometimes counterintuitive policies are better. Here we proved an approximation factor under certain conditions and also identified scenarios when grouping jobs did not necessarily result in a better objective. We also gave a post-processing algorithm to improve the objective value of the batching models, which resulted in optimal solutions for certain instances. Based on the instances we tested for both the iterative and efficient-area approaches, our assessment is that scheduling jobs similar in processing times within the same space yields good solutions. If processing times are sufficiently different, then grouping jobs together because they effectively utilize the space does not necessarily result in a lower sum of completion times. The efficient area model outperforms the iterative model both in terms of solution quality and run time.

Directions for future research are plentiful. We provide two MIP formulations to decide the assignment of jobs to batches, the iterative and efficient area model. Currently, we solve at most $N - 1$ instances for the iterative procedure and weigh the two objectives in the efficient area model equally. Possible enhancements could be to implement a binary search procedure that improves runtimes for the iterative model or tweak the weights in the multi-objective efficient area model. This study assumes that a single spatial resource of fixed dimension is available. An interesting extension would be to look at multiple workspace problems with varying area. We may be able to use ideas from variable size bin packing to design algorithms for this problem. Another area that merits investigation is to consider weights on the completion times of the jobs. If l_j is the weight on completion time for job $j \in J$, and we assign the number of jobs in the batch containing job j as a weight on its completion time, can we get similar results for our procedures? Lastly, although the results and analyses presented in this study pertain to the sum of completion times objective, the solution methods developed here can easily be applied to other objective functions of the problem.

References

1. Brucker, P., Kovalyov, M.Y., Shafransky, Y.M., Werner, F.: Batch scheduling with deadlines on parallel machines. Ann. Oper. Res. **83**, 23–40 (1998)
2. Caprace, J.D., Petcu, C., Velarde, M., Rigo, P.: Optimization of shipyard space allocation and scheduling using a heuristic algorithm. J. Mar. Sci. Technol. **18**(3), 404–417 (2013)
3. Castillo, I., Kampas, F.J., Pintér, J.D.: Solving circle packing problems by global optimization: numerical results and industrial applications. Eur. J. Oper. Res. **191**(3), 786–802 (2008)
4. Cho, K., Chung, K., Park, C., Park, J., Kim, H.: A spatial scheduling system for block painting process in shipbuilding. CIRP Ann. Manuf. Technol. **50**(1), 339–342 (2001)
5. Duin, C., Sluis, E.: On the complexity of adjacent resource scheduling. J. Sched. **9**(1), 49–62 (2006)
6. Garcia, C.J.: Optimization models and algorithms for spatial scheduling. Ph.D. thesis, Old Dominion University, Norfolk (2010)
7. Garcia, C., Rabadi, G.: A meta-raps algorithm for spatial scheduling with release times. Int. J. Plann. Sched. **1**, 19–31 (2011)
8. Garcia, C., Rabadi, G.: Exact and approximate methods for parallel multiple-area spatial scheduling with release times. OR Spectr. **35**(3), 639–657 (2013)
9. Koh, S., Logendran, R., Choi, D., Woo, S.: Spatial scheduling for shape-changing mega-blocks in a shipbuilding company. Int. J. Prod. Res. **49**(23), 7135–7149 (2011)
10. Lee, C.Y., Uzsoy, R., Martin-Vega, L.A.: Efficient algorithms for scheduling semiconductor burn-in operations. Oper. Res. **40**(4), 764–775 (1992)
11. Lee, K., Jun, K.L., Park, H.K., Hong, J.S., Lee, J.S.: Developing scheduling systems for Daewoo shipbuilding: {DAS} project. Eur. J. Oper. Res. **97**(2), 380–395 (1997)
12. Leung, J., Kelly, L., Anderson, J.H.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton (2004)
13. Lodi, A., Martello, S., Monaci, M.: Two-dimensional packing problems: a survey. Eur. J. Oper. Res. **141**(2), 241–252 (2002)
14. Mathirajan, M., Sivakumar, A.: A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. Int. J. Adv. Manuf. Technol. **29**(9–10), 990–1001 (2006)

15. Park, K., Lee, K., Park, S., Kim, S.: Modeling and solving the spatial block scheduling problem in a shipbuilding company. Comput. Ind. Eng. **30**(3), 357–364 (1996)
16. Perng, C., Lai, Y.C., Ho, Z.P.: A space allocation algorithm for minimal early and tardy costs in space scheduling. In: International Conference on New Trends in Information and Service Science, 2009 (NISS '09), pp. 33–36 (2009)
17. Pisinger, D., Sigurd, M.: The two-dimensional bin packing problem with variable bin sizes and costs. Discret. Optim. **2**(2), 154–167 (2005)
18. Raj, P., Srivastava, R.K.: Analytical and heuristic approaches for solving the spatial scheduling problem. In: 2007 IEEE International Conference on Industrial Engineering and Engineering Management, pp. 1093–1097 (2007)
19. Smith, W.E.: Various optimizers for single-stage production. Nav. Res. Logist. Q. **3**(1–2), 59–66 (1956)
20. Vazirani, V.V.: Approximation Algorithms. Springer, New York (2001)
21. Williamson, D.P., Shmoys, D.B.: The Design of Approximation Algorithms, 1st edn. Cambridge University Press, New York (2011)
22. Zhang, Z., Chen, J.: Solving the spatial scheduling problem: a two-stage approach. Int. J. Prod. Res. **50**(10), 2732–2743 (2012)
23. Zheng, J., Jiang, Z., Chen, Q., Liu, Q.: Spatial scheduling algorithm minimising makespan at block assembly shop in shipbuilding. Int. J. Prod. Res. **49**(8), 2351–2371 (2011)