Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

MULTI-OBJECTIVE OPTIMIZATION OF A SORTING SYSTEM

Arjan Smit
Jelle Adan
Patrick C. Deenen

Department of Industrial Engineering
Eindhoven University of Technology
PO Box 513
Eindhoven, 5600 MB, THE NETHERLANDS

ABSTRACT

This paper addresses the optimization of a sorting system encountered in the semiconductor industry. The system consists of parallel sorting machines and a material handler to transport materials to, from and between machines. The problem is decomposed into multiple subproblems. For each of these subproblems heuristic methods are proposed. A discrete-event simulation is used to study the performance of the system using these heuristics. Application on two real-world cases shows that this heuristic approach can significantly decrease the makespan and cost, yielding practically feasible schedules.

1 INTRODUCTION

This research is motivated by a die sorting process in the semiconductor industry. The production process of integrated circuits (ICs) starts with wafer fabrication, where many ICs are fabricated on a blank disk of semiconducting material (such as silicon) using photo-lithography techniques. Due to the nature of the production process, it is unachievable that each IC on a wafer, also called a *die*, has exactly the same electrical properties. In wafer testing, the dies are probed and categorized into so-called *classes*. Each die class is defined by limits on certain electrical properties. According to these classes, the dies are sorted into bins by designated sorting equipment. This paper focuses on the optimization of the die sorting process through the development of multiple scheduling heuristics. The reader is referred to Nishi and Doering 2007 for an elaborate overview of the semiconductor manufacturing process.

Prior to die sorting, tested wafers are stored in a warehouse. At this point, wafers lie on a thin piece of foil, a so-called *wafer frame*. From the warehouse, wafers are transported to a number of parallel sorting machines. Besides wafers, a sorting machine can contain multiple *bins*. These bins are physically similar to a wafer frame but smaller. The sorting machine picks up dies from the wafer and sorts them onto bins. A bin can only contain a single die class. Full wafers and empty bins need to be supplied to the sorting machine. Empty wafer frames and filled bins need to be removed from the machine. Similar to the wafers, bins (empty and filled) are stored in a warehouse. The movement of materials between machines and the warehouse is done by a material handler. Transport times are significant compared to the time a machine requires to sort a wafer and therefore can not be neglected. Bins are expensive and when a bin is only partially filled, the value of the sorted dies drastically decreases. Hence, for a given number of wafers to be sorted, one objective is to minimize the number of partially filled bins. Therefore, it can also be beneficial to transport partially filled bins between sorting machines, in order to further fill up the bins. Additionally, it is desired to sort the wafers as fast as possible, i.e. minimizing the makespan.

In the past, major drivers to decrease costs in the semiconductor industry included increasing wafer sizes, decreasing die sizes, and improving the yield. While this will continue, it is likely that efforts to improve operational processes become more important to realize further cost reductions (Mönch et al.

2011). Although this research is motivated by the semiconductor industry, its application and challenges to reduce costs extend to other industries. For this reason, the problem at hand is regarded in a general sense: a wafer that contains unsorted dies is referred to as a batch of unsorted items.

In this sorting problem, numerous decisions are to be made. Firstly, the unsorted batches have to be assigned to the parallel sorters. Once assigned, the sequence in which they are sorted needs to be determined. At last, the routing of the material handler needs to be determined, i.e. when are batches delivered and picked up from the sorters, when are empty bins delivered, partially filled bins transported between sorters, and finally brought back to the warehouse.

The assignment of unsorted batches to the parallel sorters resembles a parallel machine scheduling problem, a problem that is widely studied in literature. However, frequently made assumptions are that the transport times of jobs to machines are negligible compared to the process time (Lee and Chen 2001; Saidi-Mehrabad et al. 2015) and that all jobs are directly available for processing at the start (Mokotoff 2001; Fathi and Barnette 2002; Azab and Naderi 2015; Wang et al. 2018). On the other hand, when transportation times are not negligible, such as in vehicle routing problems, it is usually known up front which jobs are needed at a certain location and time (Van Woensel et al. 2008; Huang et al. 2012). The parallel machine scheduling problem with the objective to minimize the makespan alone is NP-complete and therefore it is often not possible to provide an exact optimal solution within a reasonable time limit, even for moderately sized problems. This justifies the use of heuristics, also in the current problem.

Given the complexity of the problem, it is not uncommon to hierarchically decompose the complete problem into smaller, more manageable subproblems (Crama et al. 2012). In this paper, it is proposed to decompose the sorting problem into the following subproblems: (i) batch allocation, (ii) batch sequencing and (iii) material handler routing. With the objectives to minimize the makespan while simultaneously minimizing the number of partially filled bins, heuristic methods are proposed for each subproblem. Where possible, individual heuristics are compared to mathematical programming solutions. Finally, the complete solution approach is applied to a real-world case study and a comparison is made with current practice.

The remainder of this paper is structured as follows: in the next section a detailed description of the sorting system is given. In Section 3 the proposed heuristic method is explained. Then, in Section 4 this solution method is applied a real-world case study and the results are compared to current practice. Finally, conclusions and suggestions for future research are given in Section 5.

2 PROBLEM DESCRIPTION

2.1 Sorting System

The system sorts items and each item has a specific class. Items are stored in *carriers*, such that they can be transported through the system. A carrier is either a *batch* or a *bin*, which is a carrier with items from multiple different classes or a carrier with items from one single class, respectively. Thus, a batch contains unsorted items and a bin contains sorted items. However, batches and bins can also be empty.

The main purpose of the system is to sort items based on their class by moving them from batches (of unsorted items) to bins (of sorted items). The sorting system consists of three main components, shown in Figure 1: (i) a *warehouse*, (ii) a *material handler* (also referred to as merely *handler* or (MH) and (iii) *sorting machines* (SMs). The warehouse stores both full and empty carriers. A sorting machine sorts items by moving them from a batch into designated bins. The handler transports carriers between the warehouse and sorting machines or between sorting machines themselves. Both the handler and the sorting machines have an internal buffer with a specified capacity to store batches and bins.

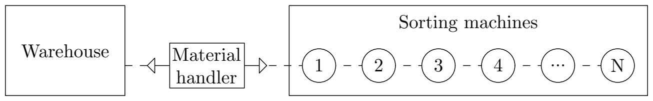


Figure 1: Schematic overview of the sorting system.

Generally, the sorting system operates in cycles. One cycle starts with feeding full batches and empty bins to the warehouse. Then, the handler transports these batches and bins to the multiple sorting machines, whereafter the sorting machines sort the items. After the sorting process, empty batches and full bins are picked up by the handler and transported back to the warehouse. Finally, the empty batches and full bins are removed from the warehouse and the next cycle begins. An overview of all parameters and the values used in the considered use cases are shown in Table 1.

2.2 Warehouse

The warehouse can store carriers and has an infinite capacity. It is assumed that there are no travel times for the handler within the warehouse.

2.3 Material handler

The handler travels with a constant speed between the warehouse and the sorting machines. If the handler is at the position of the warehouse it can immediately start picking carriers from the warehouse or placing them back into the warehouse. The time it takes to transfer one carrier is constant, and the handler's capacity for bins and batches is limited.

2.4 Sorting machines

There are multiple identical sorting machines in the system, and each sorting machine consists of several components: the internal buffer, the internal transfer unit and the work area, see Figure 2. The internal buffer has a limited capacity and both the material handler and the internal transfer unit can simultaneously access this buffer. The internal transfer unit moves one carrier at a time from the buffer to the work area and vice versa. The sorting operation happens in the work area, where merely one bin and one batch can be present. This means that items can only move from batches to bins. After an item is sorted in a bin, the item cannot be moved to another bin. Once a batch is loaded into the work area, all items have to be sorted before the empty batch can be unloaded again, i.e., the scheduling of these batches is *non-preemptive*. In contrast, bins can always be loaded and unloaded to the work area. A batch which is loaded into the work area has to be checked before it can be sorted. Furthermore, per batch, the items are sorted in groups per class. That is, after a batch is loaded, the sorting machine starts to sort all items of class A, then sorts all items of class B, and so forth. The sequence in which different classes are sorted per batch is in ascending order of the quantity of items per class, i.e. the class with fewest items is sorted first and the class with most items is sorted last.

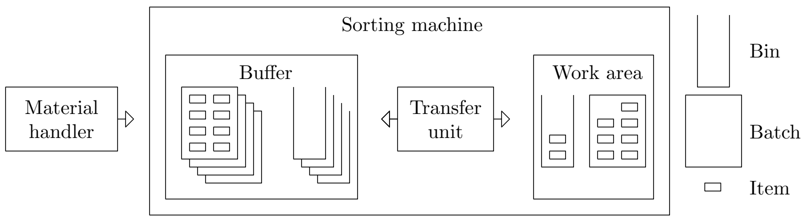


Figure 2: Schematic overview of a single sorting machine with the internal buffer, the internal transfer unit and the work area. Both the transfer unit and the material handler can access the buffer.

2.5 Objective

As mentioned before, the purpose of the system is to sort items according to their class by moving them from batches (of unsorted items) to bins (of sorted items). The system operates in cycles, where the system starts with full batches and sufficiently many empty bins and ends with empty batches and filled bins. A fully filled bin is worth significantly more than a partially filled bin and the system should finish its cycle as fast as possible. During the sorting process, a partially filled bin is referred to as a WIP bin if it still possible to fully fill the bin, given all the items which still have to be sorted, including the ones on other machines and in the warehouse. If this is not possible anymore, this partially filled bin is referred to as a deadlock-WIP (DWIP) bin. Obviously, all partially filled bins at the end of the sorting cycle are DWIP bins. Therefore, the main goal of this work is to generate a schedule σ which minimizes the objective

$$f(\sigma) = C_{max} - w \cdot \sum_{i \in C} \left[F_i - DWIP_i\right], \tag{1}$$

where C_{max} is the makespan, and F_i and $DWIP_i$ are the total amount of fully filled bins and DWIP bins, respectively, of class i. w is a weight factor and C denotes the set of all classes. The two components – minimizing C_{max} and maximizing $\sum_{i \in C}[F_i - DWIP_i]$ – in this multi-objective optimization problem are conflicting. Maximizing $\sum_{i \in C}[F_i - DWIP_i]$ would result in a schedule where batches with items of the same class are not sorted simultaneously to avoid unnecessary partially filled bins, while minimizing C_{max} would lead to schedules where batches are sorted simultaneously as much as possible. In the best case, all the items add up precisely to a multiple of C_{bin} and the total number of DWIP bins is 0. However, it is more realistic to say that in the best case each class that needs to be sorted results in one DWIP bin, and the lower bound is equal to the number of classes that is sorted into bins. An upper bound for the total number of full bins of class i is obtained by summing all the items of class i, dividing by C_{bin} and rounding down to the nearest integer.

Table 1: Parameters of the sorting system with the values used in the real-world case study.

Material handler			
Variable	Value	Unit	Description
$t_{transfer}$	50	$s/carrier$	Time to transfer carriers from the handler to the warehouse/sorting machines or vice versa.
v_{MH}	1.0	m/s	The speed of the handler.
$C_{bin,MH}$	80	-	Number of bins that can be placed in the handler's internal buffer.
$C_{batch,MH}$	20	-	Number of batches that can be placed in the handler's internal buffer.
R	0.5	-	Fill rate of the internal buffer, see Section 3.4.3.
Sorting machines			
Variable	Value	Unit	Description
n_m	20	-	Number of sorting machines.
$C_{bin,SM}$	200	-	Number of bins that can be placed in each sorting machine's internal buffer.
$C_{batch,SM}$	20	-	Number of batches that can can be placed in each sorting machine's internal buffer.
t_{sort}	1	$s/item$	Time to sort a item from a batch to a bin.
t_{load}	50	s	Time to load or unload a batch or bin to the work area of the sorting machine.
t_{check}	2000	s	Time to check a batch after the batch is loaded into the work area of the sorting machine.
General			
Variable	Value	Unit	Description
C_{bin}	20000		The capacity of a bin.
d_{SM}	5.0	m	Distance between two subsequent sorting machines.
d_W	5.0	m	Distance between the warehouse and the first sorting machine.

3 SOLUTION METHODS

Due to the complexity of the complete problem, this work proposes to decompose the complete problem described in Section 2.5, into the following subproblems, which will be discussed in separate sections.

1. Batch allocation. Which batches are allocated to which machines?

2. Batch sequencing. Per machine, in what sequence are the allocated batches sorted?
3. Bin switching. When does a machine, during the sorting process, switch to a different bin?
4. Material handler routing. Which carriers should be picked or placed at which location at what time?

It is important to note that the batch allocation and sequencing decisions are taken prior to execution, while the bin switching and material handler routing decisions are taken during execution. Each step attempts to individually optimize the overall objective function, hence, to put more focus on one of the conflicting terms, adjustments are required in multiple stages of the decomposed approach.

3.1 Batch Allocation

The first subproblem concerns the allocation of batches to a number of identical parallel sorters. The objective of this problem is previously stated in (1). At this level, movement of partially filled bins between sorters is not considered. Given this constraint, it is not necessary to consider the maximization of the number of filled bins. Consequently this term is redundant and may be omitted in the objective function. The set of machines is denoted by M, the set of unsorted batches by B and the set of classes by C. As mentioned, all machines are initially empty. As a consequence, not all machines can start processing at the start, since they need to wait before the material handler supplies them with batches and empty bins. The time at which machine i can start processing is denoted by t_i^{start} with $i \in M$. This depends on the policy under which the material handler operates, which will be explained more elaborately in Section 3.4.1. The required time to sort batch i, with the assumption that a sorter can process without interruptions and requires the minimal number of bins, is P_i with $i \in B$. The number of items of class j in batch i is given by L_{ij} with $j \in C$. Using this notation the model formulation of this subproblem is given as follows:

$$\min_{\rho,\delta} C_{max} + w \cdot \sum_{i \in C} DWIP_i, \tag{2}$$

subject to

$$\sum_{j \in M} \delta_{ij} = 1 \qquad \forall\, i \in B \tag{3}$$

$$t_j^{start} + \sum_{i \in B} \delta_{ij} P_i \leq C_{max} \qquad \forall\, j \in M \tag{4}$$

$$K\rho_{ij} \geq \sum_{k \in B} \delta_{kj} L_{ki} \qquad \forall\, j \in M, i \in C \tag{5}$$

$$\sum_{j \in M} \rho_{ij} \leq DWIP_i \qquad \forall\, i \in C \tag{6}$$

where

$$\delta_{ij} = \begin{cases} 1 & \text{if batch } i \text{ is allocated to sorter } j \\ 0 & \text{otherwise} \end{cases}$$

and

$$\rho_{ij} = \begin{cases} 1 & \text{if class } i \text{ is sorted at sorter } j \\ 0 & \text{otherwise} \end{cases}$$

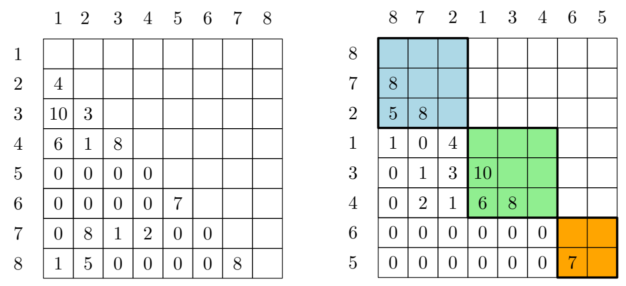
Objective function (2) minimizes number of DWIP bins and the makespan. Constraints (3) ensure that each batch is allocated to exactly one machine. Constraints (4) state that the makespan is equal to the maximum completion time. Constraints (5) state that if a class is sorted on a machine it always results in one DWIP bin, where K is a large integer. Note that the rare case where all bins can be filled precisely on a machine is neglected. At last, constraints (6) determines the total number of DWIP bins of a specific class over all machines. In Sections 3.1.1 and 3.1.2 two solution methods are described to solve the ILP.

3.1.1 Gurobi Optimizer

Gurobi Optimizer is a commercial optimization solver for mathematical programming problems, such as the integer linear programming (ILP) problem introduced in Section 3.1. For a more elaborate description of this solver the reader is referred to (Gurobi Optimization. 2019).

3.1.2 Clustering Algorithm

Given the size and complexity of the problem at hand, it is not possible to find an optimal solution within a reasonable time limit. For this reason, a heuristic method is proposed as alternative. The proposed algorithm attempts to cluster batches with the same classes of items on the same sorter as much as possible. First, a similarity matrix is constructed for the unsorted batches. The similarity between two batches is simply the number of classes they have in common. Subsequently, the matrix is clustered using the Ward variance minimization algorithm (Murtagh and Legendre 2014). An example is provided in Figure 3.



(a)

	1	2	3	4	5	6	7	8
1								
2	4							
3	10	3						
4	6	1	8					
5	0	0	0	0				
6	0	0	0	0	7			
7	0	8	1	2	0	0		
8	1	5	0	0	0	0	8	

(b)

	8	7	2	1	3	4	6	5
8								
7	8							
2	5	8						
1	1	0	4					
3	0	1	3	10				
4	0	2	1	6	8			
6	0	0	0	0	0	0		
5	0	0	0	0	0	0	7	

Figure 3: Example of (a) the initial unclustered similarity matrix and (b) the clustered similarity matrix for eight batches. Clusters are indicated with bold colored squares.

Next, a lower and an upper bound for the makespan are determined. Determination of the lower bound is rather straightforward:

$$C_{lower} = \frac{\sum_{i \in M} t_i^{start} + \sum_{i \in B} P_i}{|M|} \tag{7}$$

i.e. through summation of the batch process times and the times at which the sorters can start, and division by the total number of machines. To determine the upper bound, a greedy solution is determined. This is done by first ordering the batches, disregarding the clusters, by decreasing processing time. According to this sequence, the batches are consecutively allocated to a machine in a greedy manner, i.e. to the sorter with the lowest completion time. The value of the reduced objective function (2) for this solution is denoted by f_{greedy}. Most ideally, the DWIP term is equal to the total number of classes, i.e. one DWIP bin for every class. Therefore, an upper bound for the makespan is given by:

$$C_{upper} = f_{greedy} - w \cdot |C| \tag{8}$$

Now, using the upper and lower bounds, a solution based on the similarity matrix can be derived. First, a bound C_{bound} is set equal to the lower bound C_{lower}. The set of machines is ordered with the lowest t_i^{start} first. The batches are allocated to the machines cluster-by-cluster using the Next Fit (NF) algorithm (Hofri 1984). According to the NF algorithm, the first batch from the first cluster is allocated to the first machine where it fits, i.e. its completion time does not surpass C_{bound}. If this is not the case, it is attempted to allocate the batch to the next machine. The next batch is allocated to the first machine where it fits, starting at the machine where the last batch was allocated. This way, batches in the same cluster are allocated to the same machine as much as possible. This procedure is schematically depicted in Figure 4. If it is not possible to allocate all batches while satisfying the bound, then C_{bound} is increased by $(C_{upper} - C_{lower})/s$, where s specifies the number of steps. This process continues until a solution is found or $C_{bound} > C_{upper}$.

In the latter case, the schedule is definitely worse than the greedy solution. The number of sequences in which the clusters can be allocated to the machines increases as a factorial. Therefore it is not feasible to search through all sequences. To limit computational effort but explore the search space further, the procedure is repeated taking each cluster as a start. At last, a final attempt to improve the solution is made by grouping the batches by machine assignment. The groups are successively reallocated, starting with the group that has the largest total processing time. This group is allocated to the machine with the earliest t_i^{start}, then the group with the second largest total processing time and so on, until all groups are reallocated.

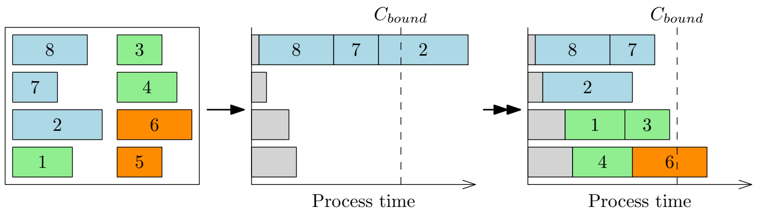


Figure 4: Illustrative example of the NF algorithm. The numbers indicate different batches and the colors correspond to the clusters in Figure 3b.

3.2 Batch sequencing

Now that the batches are allocated to the sorters, the sequence in which the batches are processed at an individual sorter needs to be determined. A simple heuristic is applied to make this decision: for each sorter, the assigned batches are ranked in ascending order according to the number of bins that are required per unit of process time. The reasoning behind this is as follows: in the beginning all machines are idle, waiting to be supplied by the material handler. With this heuristic, it is attempted to minimize the required supply of a sorter in the beginning, such that the material handler can focus on serving all idle machines.

3.3 Bin switching

At this point, the batches have been allocated to the machines, the sequence in which batches are sorted has been determined and it is given that the sequence of which classes are picked in a batch is in ascending order based on the quantity of items per class. Now, it has to be determined when a machine should switch bins during the sorting process. Obviously, a machine has to switch bins if (i) it reached its maximum bin capacity or (ii) it starts to sort items from another class. However, it might also be preferable to switch before one of the aforementioned conditions is met. This is illustrated in Figure 5, where both machine 1 and 2 are sorting items from the same class. In the left schedule both machines switch bins after the maximum capacity of the bins is reached, resulting in using in total 4 bins (2 fully filled and 2 DWIP bins). In the right schedule machine 1 switches bins earlier, such that exact enough items remain to fully fill bin 2 and bin 1 can be moved to machine 2. The time it has to transfer the bin from machine 1 to 2, is also shown in Figure 5. The right schedule results in using a total of 3 (2 fully filled bins and 1 DWIP bin). As is illustrated with this example, the number of resulting DWIP bins can be influenced by transferring WIP bins between machines. Therefore, a heuristic is proposed to determine when a sorting machine has to switch bins during the sorting process.

A WIP bin which is not going to be filled anymore at the current machine is referred to as a *transfer bin*. This transfer bin can be caused by two reasons: (i) there are no items going to be sorted anymore on that machine from the corresponding class or (ii) the bin switching heuristic determined that this bin is not going to be filled anymore. The bin switching heuristic starts with estimating the minimum number of transfer bins needed for class c in machine m by

$$TB_c = \frac{\sum_{m \in M} N_{rem,c,m}}{C_{bin}} \quad \forall \ c \in C \tag{9}$$

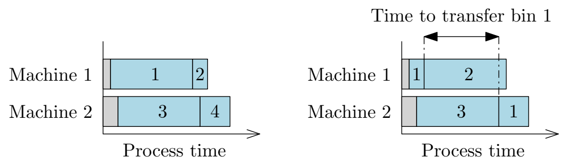


Figure 5: Illustrative example of the possible performance gain with bin switching. All bins are from the same class and, contrary to Figure 4, the numbers indicate different bins.

where TB_c is the number of transfer bins needed for class c. $N_{rem,c,m}$ is the number of remaining items of class c at machine m that will be sorted into the bin that cannot be filled totally. This is calculated by

$$N_{rem_{c,m}} = (N_{sort_{c,m}} + N_{WIP_{c,m}} + N_{WIP,MH_c}) \mod C_{bin} \;\; \forall \;\; c \in C, \forall \;\; m \in M \tag{10}$$

where $N_{sort,c,m}$ is the total number of items of class c which are still going to be picked (including items from batches which are still in the warehouse) at sorting machine m. $N_{WIP_{c,m}}$ is the total number of items of class c in WIP bins (which are not transfer bins) present at machine m and N_{WIP,MH_c} is the total number of items of class c in WIP bins in the material handler which are en route to machine m.

The bin switching heuristic works as follows. Before filling a bin, TB_c is computed with the current state of the system. Then, it is checked whether the total amount of transfer bins of class c is smaller than TB_c. If this is true, the bin will become a transfer bin and is filled with $N_{rem_{c,m}}$. If this is false, the bin will be filled as far as possible. Once the bin is filled until the determined amount, the SM has to load a new bin. The policy used for this is as follows: if there is a WIP bin available of the class which is going to be sorted next, this WIP bin is loaded. If there are multiple WIP bins available, the WIP bin with the highest quantity is chosen. If there are no WIP bins available, an empty bin is loaded. Finally, if there are also no empty bins available, the SM will be idle until it is supplied with new bins by the material handler.

3.4 Material handler routing

A heuristic is proposed which determines, depending on the state of the system, what the material handler has to do. The material handler can be in three different *modes*: *Supply, WIP* and *Clear*, which is managed by Algorithm 1. t is the current time in the system and $t_{idle,m}$ is the time that the sorting machine will be idle, given that it will receive no new bins or batches. A summary of the modes will be given here, after which each of the modes will be explained elaborately in following sections. The system begins in Supply mode, in which the material handler will supply the sorting machines with carriers by moving between the warehouse and sorting machines. The MH continues doing this, until all machines are able to keep sorting (without being supplied by the handler) for at least threshold time τ. Then, the material handler switches to WIP mode, in which it will move WIP bins (also referred to as transfer bins) between machines. If the MH could not execute a task in one of the aforementioned modes, then it switches to Clear mode. In this mode, the handler returns full bins and empty batches by moving them from machines to the warehouse. If the handler could not execute any task in the Clear mode, i.e., no carriers were ready to be returned, the material handler has no task to do and will be idle until the next event triggered by one of the machines.

3.4.1 Supply mode

In this mode, the material handler will supply the sorting machines with carriers from the warehouse. Two different policies are used: *single loading policy* and the *multi loading policy*, where one sorting machine is supplied simultaneously or multiple sorting machines are supplied simultaneously, respectively. For these policies an estimate of the makespan for machine m at time t, $\hat{C}_{max,m}(t)$, is calculated. $\hat{C}_{max,m}(t)$ is estimated under the assumption that machine m can continuously process, i.e. the supply of batches and empty bins is sufficient to keep sorting. Knowing the state of the system at time t, it is known how many items each sorting machine m still has to pick and $\hat{C}_{max,m}(t)$ can be computed.

Algorithm 1 Heuristic to determine the next task of the material handler.

```
 1: if $\min_{m \in M}[t_{idle,m}] \leq t + \tau$ then
 2:     Supply mode                                        ▷ See Subsection 3.4.1
 3: else
 4:     WIP mode                                           ▷ See Subsection 3.4.2
 5: end if
 6: if no task executed then
 7:     Clear mode                                         ▷ See Subsection 3.4.3
 8: end if
 9: if no task executed then
10:     $t \leftarrow$ time of next event
11: end if
```

Under the single loading policy, sorting machine m is selected with the highest $\hat{C}_{max,m}(t)$. The material handler starts loading carriers until the condition $t_{idle,m} > t + \tau$ is met. If this condition is met, the material handler departs from the warehouse to deliver carriers to this machine.

The multi loading policy starts in the same way as the single loading policy by selecting machine m with the highest $\hat{C}_{max,m}(t)$. Then it starts to pick one carrier from the warehouse for machine m. After each pick, it calculates how long it would take the MH to deliver all the carriers and travel back to the warehouse. This time denoted as $\tau_{deliver}$, and then $\max_{m \in M}[\hat{C}_{max,m}(t + \tau_{deliver})]$ is computed. If the condition

$$\max_{m \in M}[\hat{C}_{max,m}(t + \tau_{deliver})] > \max_{m \in M}[\hat{C}_{max,m}(t)] \tag{11}$$

is true, then another carrier is picked for machine m which has the largest $\hat{C}_{max,m}(t + \tau_{deliver})$. $\tau_{deliver}$ is computed again and condition (11) is checked again. This is repeated until condition (11) no longer holds or if the maximum capacity of either the material handler or one of the sorting machines is reached. Then, the material handler departs from the warehouse and starts supplying the machines.

3.4.2 WIP mode

For this mode the time t_m is introduced. t_m is the time that SM m will take an empty bin to fill. Before t_m, the SM is filling WIP bins. In the WIP mode, the algorithm checks which transfer bins each SM $m \in M$ requires, based on the classes that still have to be sorted on this SM. The MH starts with serving the sorting machine with the earliest time t_m. It takes transfer bins from the nearest other SM and transfers these to this SM. This transfer has to satisfy the condition that the transfer bin will be delivered on time, i.e, before t_m. If this condition is not met, it iterates over the machine until a feasible transfer is found.

3.4.3 Clear mode

In the Clear mode, the MH picks carriers that have to be returned from the closest SM, with respect to the current location of the MH. The MH continues doing this until either R (chosen to be 50% in this work) of the MH's internal buffer is filled with these returning carriers or there are no returning carriers in the sorting machines left. Then, the MH moves to the warehouse and unloads these carriers.

3.5 Discrete-event simulation

Using the heuristics previously explained, a discrete-event simulation (DES) is executed to obtain the complete schedule and analyze the performance of the system. Prior to the first event in the DES, the simulation is initialized by running the batch allocation (Section 3.1) and the batch sequencing (Section 3.2) heuristic. After that, the events of the material handler are handled by the material handler routing heuristic

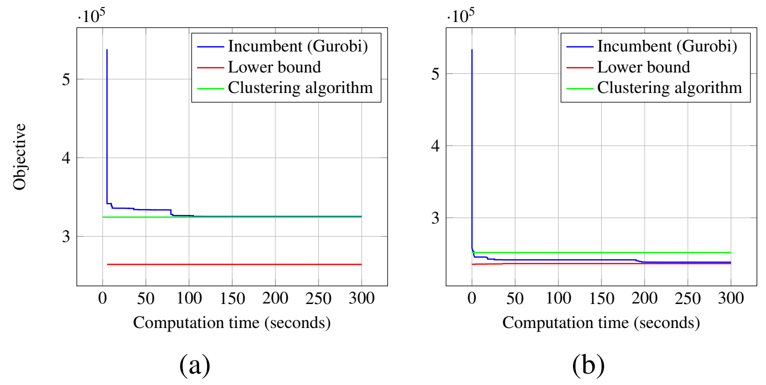


Figure 6: A comparison between Gurobi and the clustering algorithm for (a) Case 1 and (b) Case 2.

(Section 3.4) and the events of the sorting machines by the bin switching heuristic (Section 3.3). At every event, where either a SM or the MH has to make decisions, the corresponding heuristic is executed.

4 REAL-WORLD CASE STUDY

In this section, the proposed solution method is applied to real-world cases. All algorithms are coded in Python 3.7 and all experiments are run on a computer with an Intel Core i7-5700HQ processor running at 2.70 GHz and 8 GB of RAM memory. For all cases the weight factor w in the objective function (Equation 2) is set to 20, the number of steps s in the clustering algorithm is set to 100 and the threshold time τ in Algorithm 1 is set to 4100 seconds.

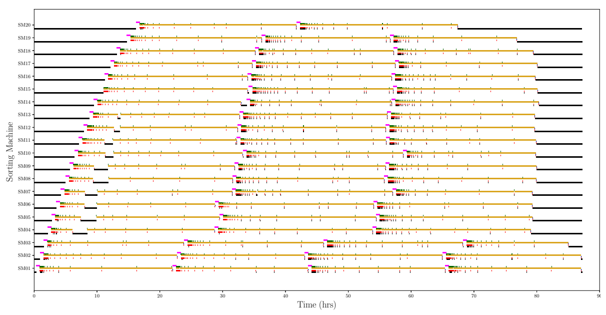
4.1 Cases

Two real-world cases are obtained from an industrial partner, which represent two extreme scenarios that are both frequently encountered in practice:

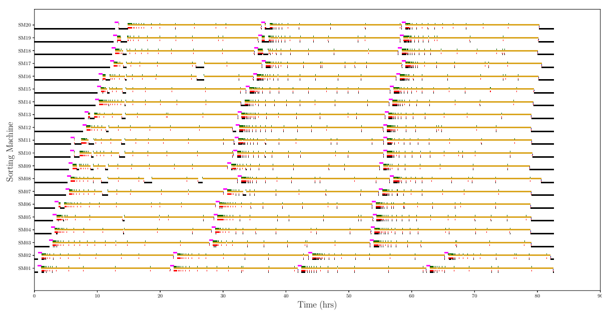
- Case 1 contains 62 unsorted batches with in total 190 different classes. On average a batch contains 78305 items, with a standard deviation of 4370. Each batch contains an average of 28 different classes, varying between 23 and 38. All items are required to be sorted into bins.
- Case 2 contains 58 unsorted batches with in total 4 different classes. On average a batch contains 393875 items, with a standard deviation of 8805, varying between 38630 and 121680. Each batch contains items of all four classes. The most abundant class does not have to be sorted into a bin and remains on the batch carrier. In this case, this is always the same class. As a result, the process time of a batch varies significantly.

4.2 Results

Prior to solving the cases entirely, the batch allocation problem is solved using the two solution methods presented in Sections 3.1.1 and 3.1.2, i.e. the commercial mathematical programming solver Gurobi and the proposed clustering algorithm respectively. The results obtained with both methods are depicted in Figure 6. Gurobi did not prove optimality for any of the cases, despite running for several hours. In Case 1 the clustering algorithm found the best solution, while in Case 2 Gurobi found the best solution. The gap between the two methods with respect to the heuristic is 0.24% and -5.34% for Case 1 and 2 respectively. This comparison proves that the heuristic method is able to provide descent solutions to the batch allocation problem within minimal required computational time. In practice, the simplicity of the heuristic is preferred over an expensive commercial solver. Even though better solutions may obtained in some cases, this only concerns the first subproblem, and does not translate directly to the overall objective.



(a)



(b)

Figure 7: Case 1 schedules with (a) single and (b) multi loading policy. Colors indicate idle time (black), sorting (orange), bin unloading (red), bin loading (green), batch loading (pink) and unloading (blue).

Next, both cases are provided as input to the DES in order to determine the performance measures of the system. A comparison is made between current practice and the novel heuristic, with two different routing policies. The results for Case 1 and 2 are summarized in Table 2. Firstly, it can be seen that the novel heuristic significantly reduces the number of DWIP bins and increases the amount of full bins when compared to current practice. Despite the fact that makespan and the number of fully filled bins are conflicting objectives, this is achieved at a makespan that is approximately equal or even lower than current practice. Although the computational time required for the current method is extremely low, the time required for the proposed heuristic is also acceptable in practice.

In Table 2 it can be seen that the multi loading heuristic is able to reduce the makespan further compared to the single loading policy. The difference between the two routing policies is most clear in a visualization of the schedules. For Case 1 the schedules are shown in Figure 7 for both policies. Current practice refers to a simple dispatching heuristic that is currently applied, and does not consider transfer bins. When the single loading policy is applied, the first sorter is supplied, after which the material handler returns to the warehouse, before the second sorter is supplied, and so on. This explains the linear increase of the idle time at the beginning of the schedule from SM01 to SM20. When the multi loading policy is applied, the increase is not exactly linear. Additionally, the total idle time at the beginning is equal or less when compared to the single loading policy. This enables a reduction in the makespan.

For Case 1 and 2, the minimum total number of DWIP bins is 190 and 3, and the upper bound for the total number of full bins is 161 and 224, respectively. For Case 2, the single loading policy comes very close, while the multi loading heuristic is able to match the upper bound for the number of full bins. For Case 1, the gap is significantly larger, which is likely due to the large number of different classes per batch.

Table 2: Performance measures for the novel heuristic compared to current practice.

Case 1					
Method	Routing policy	Makespan (hrs)	DWIP bins	Full bins	Comp. time (s)
Current practice	-	93.9	1526	81	2
Novel heuristic	Single loading	87.0	854	102	30
Novel heuristic	Multi loading	82.5	787	106	74
Case 2					
Method	Routing policy	Makespan (hrs)	DWIP bins	Full bins	Comp. time (s)
Current practice	-	87.0	60	198	1
Novel heuristic	Single loading	69.8	7	221	18
Novel heuristic	Multi loading	69.3	4	224	17

5 CONCLUSIONS AND FUTURE WORK

In this paper the entire sorting problem was decomposed into subproblems for which several heuristics were proposed. Application to two representative real-world cases shows that the heuristic approach performs significantly better than current practice. Though this is promising, further investigation is required to gain more confidence about the performance of the proposed solution method. Also, it is relevant to investigate which properties of the sorting system (e.g. the speed of the material handler, carrier buffer sizes) are most critical to the system performance.

REFERENCES

Azab, A., and B. Naderi. 2015. "Modelling the problem of production scheduling for reconfigurable manufacturing systems". *Procedia CIRP* 33(0):76–80.

Crama, Y., A. G. Oerlemans, and F. C. Spieksma. 2012. *Production planning in automated manufacturing*. Springer Science & Business Media.

Fathi, Y., and K. Barnette. 2002. "Heuristic procedures for the parallel machine problem with tool switches". *International Journal of Production Research* 40(1):151–164.

Gurobi Optimization. 2019. *Gurobi optimizer reference manual*. http://www.gurobi.com, accessed 12th March.

Hofri, M. 1984. "A Probabilistic analysis of the Next-Fit bin packing algorithm". *Journal of Algorithms* 5(4):547–556.

Huang, Y., C. Shi, L. Zhao, and T. Van Woensel. 2012. "A study on carbon reduction in the vehicle routing problem with simultaneous pickups and deliveries". In *Proceedings of 2012 IEEE International Conference on Service Operations and Logistics, and Informatics*, 302–307. IEEE.

Lee, C.-Y., and Z.-L. Chen. 2001. "Machine scheduling with transportation considerations". *Journal of Scheduling* 4(1):3–24.

Mokotoff, E. 2001. "Parallel machine scheduling problems: A survey". *Asia-Pacific Journal of Operational Research* 18(2):193.

Mönch, L., J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose. 2011. "A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations". *Journal of Scheduling* 14:583–599.

Murtagh, F., and P. Legendre. 2014. "Ward's hierarchical agglomerative clustering method: which algorithms implement Ward's criterion?". *Journal of classification* 31(3):274–295.

Nishi, Y., and R. Doering. 2007. *Handbook of semiconductor manufacturing technology*. CRC Press.

Saidi-Mehrabad, M., S. Dehnavi-Arani, F. Evazabadian, and V. Mahmoodian. 2015. "An Ant Colony Algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs". *Computers Industrial Engineering* 86:2 – 13.

Van Woensel, T., L. Kerbache, H. Peremans, and N. Vandaele. 2008. "Vehicle routing with dynamic travel times: A queueing approach". *European journal of operational research* 186(3):990–1007.

Wang, S., X. Wang, J. Yu, S. Ma, and M. Liu. 2018. "Bi-objective identical parallel machine scheduling to minimize total energy consumption and makespan". *Journal of Cleaner Production* 193:424–440.

AUTHOR BIOGRAPHIES

ARJAN SMIT M.Sc. is an operational coördinator at ASML. He conducted this research as part of his Master's thesis within the Department of Mechanical Engineering of the Eindhoven University of Technology. His current research interests are in the area of modeling, design and optimization of manufacturing systems. His email address is asmit471@gmail.com.

JELLE ADAN is a doctoral candidate at the Department of Industrial Engineering of the Eindhoven University of Technology and a Sr. Business Process Analyst at Nexperia, Equipment and Automation Technologies (E&A). His current research interests are supply chain, manufacturing and chemical process optimization, and data mining. His email address is jelle.adan@protonmail.com.

PATRICK C. DEENEN is a doctoral candidate at the Department of Industrial Engineering of the Eindhoven University of Technology and a Sr. Business Process Analyst at Nexperia. His current research interests are in the area of modeling, control and optimization of manufacturing systems. His email address is patrickdeenen@hotmail.com.