Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, eds.

WAFER-TO-ORDER ALLOCATION IN SEMICONDUCTOR BACK-END PRODUCTION

Patrick C. Deenen
Jelle Adan
Joep Stokkermans

Industrial Technology and Engineering Centre
Nexperia
Jonkerbosplein 52
Nijmegen, 6534 AB, THE NETHERLANDS

Ivo J.B.F. Adan
Alp Akcay

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

ABSTRACT

This paper discusses the development of an efficient algorithm that minimizes overproduction in the allocation of wafers to customer orders prior to assembly at a semiconductor production facility. This study is motivated by and tested at Nexperia's assembly and test facilities, but its potential applications extend to many manufacturers in the semiconductor industry. Inspired by the classic bin covering problem, the wafer allocation problem is formulated as an integer linear program (ILP). A novel heuristic is proposed, referred to as the multi-start swap algorithm, which is compared to current practice, other existing heuristics and benchmarked with a commercial optimization solver. Experiments with real-world data sets show that the proposed solution method significantly outperforms current practice and other existing heuristics, and that the overall performance is generally close to optimal. Furthermore, some data processing steps and heuristics are presented to make the ILP applicable to real-world applications.

1 INTRODUCTION

This work is motivated by the problem of allocating semiconductor wafers to customer orders in the back-end production process of integrated circuits (ICs) at Nexperia's assembly and test facilities. The back-end sites work with a make-to-order policy, in which customer orders have to be made with wafers which are stored in a warehouse waiting to be assembled. It is a challenging task to allocate these wafers to the orders prior to assembly. Inefficient allocation can cause more produced finished goods than requested by the customer orders, i.e. overproduction. This overproduction has a severe impact on the company's profit, since it will consume more expensive material, produce excess inventory and occupy manufacturing machines longer. Thus, in the $470.3 billion semiconductor industry (Association. 2019), which is still growing every year, improving the wafer allocation process to minimize overproduction, is highly valuable.

The manufacturing process of ICs can basically be divided into the following four steps. The first one is wafer fabrication, where many ICs are fabricated on a blank disk of semiconducting material (such as silicon) using photo-lithography techniques. A single IC on a wafer is called a *die*. The second step is wafer testing, where the dies are probed on their electrical properties and can be identified as a good or bad die. The first two steps, wafer fabrication and wafer testing, are carried out in the front-end *wafer fabs*, whereafter the wafers are transported to the back-end assembly and test facilities for the third step. Here, the good dies are cut out of the wafers and assembled into a package. In the fourth and final step, these packaged ICs receive a full final functional test after which the ones that pass the tests can be sold to customers. The reader is referred to Frederix (1996) for an elaborate overview of the manufacturing process.

At the back-end facilities wafers are allocated to customer orders. This process is prone to overproduction because (1) underallocation is not allowed and (2) once a wafer is allocated to an order the complete wafer

has to be consumed. This task of allocating wafers to customer orders prior to assembly is called the *wafer allocation problem*. This problem has received significant attention by researchers and is also known as the lot-to-order matching problem (Knutson et al. 1999). The main objective is to minimize overproduction while fulfilling a given set of orders. Important to note is that the wafer allocation problem, which is a variation of the *bin covering problem*, 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 (Garey and Johnson 1990; Woeginger and Zhang 1999). This justifies the use of heuristics.

The bin covering problem is a dual version of the bin packing problem and it belongs to the packing problems. The literature on packing problems (which includes bin packing, bin covering and knapsack problems) is very broad and one can find many heuristics to approximately solve the problem. A comparison between heuristics has been made by Bischoff and Marriott (1990). The main conclusion of this comparison is that the performance of these heuristics is very domain-dependent, and their performance may vary with the number and relative size of the different items. Especially when the number of bins (orders) is small and the average size of the items (wafers) is large, it is more difficult to pack each bin effectively. More recently, Coffman Jr. et al. (2013) give a survey and classification of many approximation algorithms for the bin packing problem and showed that a First-Fit-Decreasing (FFD) heuristic has one of the best performance with respect to the worst case as well as the average case behavior for the bin packing problem, outperforming a First-In-First-Out (FIFO) policy significantly. The First-Fit-Decreasing (FFD) policy originates from the bin packing problem, as is described in Dyckhoff (1990). Knutson et al. (1999) first used the FFD algorithm for the application of matching wafer lots to customer orders. Fowler et al. (2000) and Carlyle et al. (2001) introduced several variations of the FFD policy for the same problem, where FFD1(S_i), which is an algorithm that combines a search routine with a priority rule system, eventually had the best average performance. Boushell et al. (2008) extended the FIFO and FFD algorithms with the improved endgame (IEG) heuristic, of which FIFO/IEG appears to perform the best.

In this paper a novel multi-start swap (MS-Swap) heuristic is proposed to solve the wafer allocation problem. MS-Swap uses multiple initial solutions complemented with a smart swapping mechanism. This algorithm is applied to real-world problem instances. Although the problem in this paper is motivated by Nexperia, its potential applications extend to many manufacturers in the semiconductor industry. Furthermore, the performance of MS-Swap is compared to the current practice at Nexperia as well as to the best existing methods, i.e. FFD/IEG and FIFO/IEG. Also, a mathematical programming solver is used to serve as a benchmark for the selected problem instances and gain insight in the performance of all solution methods.

The remainder of this paper is organized as follows: in Section 2 the mathematical problem description is given. The solution methods used in this work are described in Section 3. Details on the application are explained in Section 4 and the results are presented in Section 5. Finally, conclusions and suggestions for future work are given in Section 6.

2 PROBLEM DESCRIPTION

The wafer allocation problem (WAP) has many similarities to the well-known *variable-sized bin covering problem* (VSBCP), which in turn is a variation of the *classic bin covering problem*. The classic bin covering problem is the dual version of the *bin packing problem*. As is done by Assmann et al. 1984, it can be described as follows: suppose that there is a set of *items* $I = \{a_1, a_2, \ldots, a_n\}$, each having a *size* $s(a_i) > 0$, and an infinite number of equally sized bins, each having a *threshold* capacity $T > 0$. The objective is to determine the maximum number m such that I can be partitioned into sets $I_1, ..., I_m$ where each set I_i has total size $s(I_i) = \sum_{\alpha \in I_i} s(a_i) \geq T$.

The VSBCP has a finite set of variable sized bins $B = \{b_1, b_2, ..., b_m\}$ with *size* $s(b_i) > 0$ for each bin b_i, where the cumulative size of all bins is bigger than the cumulative size of all items. Similarly the objective is to cover, with the given list of items, a set of bins with the largest total size.

The bins and items in the classic bin covering problem are considered to be orders and wafers respectively in the wafer allocation problem. The set of orders is denoted by O and the set of wafers by W. The size of order i is S_i with $i \in O$ and the size of wafer j is L_j with $j \in W$, both expressed in die quantities. Using this notation the model formulation of the WAP is given as follows:

$$\min_{\delta} \sum_{i \in O} \left[\sum_{j \in W} \delta_{ij} L_j - S_i \right], \tag{1}$$

subject to

$$\sum_{j \in W} \delta_{ij} L_j \geq S_i \qquad \forall\, i \in O \tag{2}$$

$$\sum_{i \in O} \delta_{ij} \leq 1 \qquad \forall\, j \in W \tag{3}$$

where

$$\delta_{ij} = \begin{cases} 1 & \text{if wafer } j \text{ allocated to order } i \\ 0 & \text{otherwise.} \end{cases}$$

The objective function (1) minimizes the cumulative overallocation. Constraints (2) ensure that all orders are covered and constraints (3) ensure that a wafer can only be allocated to at most one order.

Generally, the set of orders O is selected such that there is sufficient supply of wafers to cover all orders. However, for real-world instances this cannot be ensured beforehand and therefore the problem can still be infeasible. If it is infeasible, one or more orders will be removed from the set of orders until the problem becomes feasible. This will be discussed more elaborately in Section 4.1.

3 SOLUTION METHODS

In this section, several methods will be described to solve the wafer allocation problem. In Sections 3.1 and 3.2, two existing methods are described briefly. For an elaborate step-by-step explanation of both these methods, the reader is referred to Boushell et al. (2008). In Section 3.3, the novel multi-start swap is described in more detail. Finally, the commercial solver used for benchmarking is discussed shortly.

3.1 First-Fit-Decreasing with Improved Endgame

The FFD/IEG algorithm consists of two parts. The first part fills the orders according to the FFD policy and the second part is the improved endgame which attempts to cover the orders in an efficient way. For the first part, the orders and wafers are sorted in non-increasing size, as in the FFD policy. The algorithm starts filling the largest order first and it only moves to the next order once the current one is covered. An order is sequentially filled with the largest available wafers. Before adding each wafer, it is first checked whether the two largest wafers can cover the order. If this is possible, the algorithm enters the so-called improved endgame, otherwise the next largest wafer is allocated to this order.

In the improved endgame (IEG), the algorithm searches for the two wafers which (1) cover the order and (2) minimize the overallocation. In our implementation, the IEG enumerates over all possibilities, unless a wafer pair is found which perfectly covers the order (i.e. zero overallocated dies).

3.2 First-In-First-Out with Improved Endgame

The FIFO/IEG is very similar to the FFD/IEG algorithm and uses the same improved endgame. The only difference is that it fills the orders with wafers in FIFO order instead of in FFD order, i.e. the wafer with first due date is allocated first instead of the largest wafer. Due to the randomness of wafer arrivals in the inventory, this results in a larger diversity of wafer sizes once the algorithm reaches the improved endgame.

This may result in improved performance (Boushell et al. 2008). Another advantage of allocating the wafers in FIFO order is that it helps to prevent wafers stocks to expire.

3.3 Multi-Start Swap Algorithm

The main procedure of the multi-start swap algorithm proposed here is outlined in Algorithm 1 below. It starts with the generation of the set of initial solutions, S_{all}, followed by attempts to find improvements for each initial solution $s \in S_{all}$ via iterations over local search (LocalSearch(s)) and 0-cost moves (ZeroCostMoves(s)). The latter search methods are both based on swap moves, i.e. the exchange of two wafers. The number of times the algorithm iterates over the sequential local search and 0-cost moves, is determined by parameter R. If the resulting objective of solution s (ObjectiveOf(s)) is smaller than the current best objective (O_{best}), the solution s and its objective are saved. In the subsequent sections the components of the proposed MS-Swap algorithm are described in more detail.

Algorithm 1 Multi-Start Swap (MS-Swap) algorithm.

$O_{best} \leftarrow \infty$
$S_{all} \leftarrow$ GenerateInitialSolutions(); ▷ See Algorithm 2

for each solution $s \in S_{all}$ **do**
    $i_{reps} \leftarrow 0$;
    while $i_{reps} < R$ **do**
        LocalSearch(s); ▷ See Algorithm 3
        ZeroCostMoves(s); ▷ See Algorithm 5
        $i_{reps} \leftarrow i_{reps} + 1$;
    end while

    if ObjectiveOf(s) $< O_{best}$ **then** ▷ ObjectiveOf(s) calculates the objective of solution s
        $O_{best} \leftarrow$ ObjectiveOf(s);
        $s_{best} \leftarrow s$;
    end if
end for

3.3.1 Initialization and Random Fill

Inherently, swap moves do not alter the number of wafers allocated to each order as well as the number of unallocated wafers. Thus, the number of wafers allocated to each order is dictated by the initial solution. Consequently, since an optimal solution contains a certain number of wafers for each order, it may not be reached from certain initial solutions. For this reason, multiple initial solutions are generated, each with a different number of wafers per order. Given the required die quantity of an order together with the available wafers, it is rather straightforward to determine the minimum and maximum number of allocated wafers for each order. The total number of initial solutions with a different number of wafers for each order then equals,

$$\prod_{i=0}^{i=I} (|W|_i^{max} - |W|_i^{min})$$

where $|W|_i^{min}$ and $|W|_i^{max}$ are the minimum and maximum number of allocated wafers to order i respectively. When the variation of die quantity among wafers is relatively small, this number remains small as well. However, since this is not necessarily the case for every problem, it is not scalable to evaluate all possible

Algorithm 2 Initialization for MS-Swap.

```
1:  procedure GENERATEINITIALSOLUTIONS
2:
3:      c_min ← MinConfiguration(|W|^min);      ▷ Configuration c_min uses min. amount of wafers per order
4:      s_min ← RandomAllocateWafers(c_min);                          ▷ Solution s_min is generated
5:      Add s_min to set S_all;                                       ▷ S_all is set of all initial solutions
6:
7:      for each order i ∈ O_all do
8:          k_inc ← 0;
9:          while k_inc < K do
10:             c_new ← Increase order i in configuration c_min with k_inc;
11:             if c_new is feasible then
12:                 s ← RandomAllocateWafers(c_new); ▷ Allocates wafers according to configuration c_new
13:                 Add s to set S_all
14:             end if
15:             k_inc ← k_inc + 1;
16:         end while
17:     end for
18:
19:     return S_all
```

configurations. Instead, the set of initial solutions, S_{all}, is generated as in Algorithm 2. First, the configuration with a minimum number of wafers for each order, c_{min}, is determined. Then, solution s_{min} is generated through a random assignment of a certain number of wafers per order, defined by c_{min}. After that, another series of initial solutions is generated where the allocated number of wafers is increased by 1 for each order separately. This procedure continues until a certain number of increments is reached, specified by a parameter K. Each newly generated configuration c_{new} is checked on feasibility, i.e., the total number of allocated wafers may not exceed the total number of available wafers. Furthermore, due to the random assignment of wafers it can happen that the initial solutions contain orders which are not filled. However, the subsequent local search methods will attempt to solve this.

3.3.2 Swap Moves

Once a random initial solution is generated, attempts are made to find improvements by means of swap moves. As mentioned, a swap move is based on the exchange of two wafers. The swap neighborhood is searched as described in Algorithm 3. The search propagates through each order as follows: at first, an allocated wafer w_i is selected. Then, swap moves with all unallocated wafers, $w_j \in W_{UA}$, are evaluated. Finally, the move that results in the largest improvement is accepted, i.e. steepest descent. This move is implemented by ImplementSwap(BestMove). Among preliminary experiments with different acceptance strategies, e.g. any descent, the strategy of steepest descent yielded the best results in terms of computational efficiency. The search for feasible swap moves continues until no more improvements are found. Then, the search continues with the next order.

The acceptance criterion for a swap move is implemented in DetermineIfImprovement, which is shown in Algorithm 4. The total allocated quantity can be smaller or larger than the total required quantity, i.e. underallocation or overallocation respectively. In case of underallocation the following two conditions have to be satisfied: (1) the move has to decrease the underallocation and (2) the swap move has to be better than the currently best found swap move (BestSwap). In case of overallocation, the swap move between two wafers, w_i and w_j, has to satisfy following three conditions: (1) the swap move has to decrease the

overallocation, (2) the swap move has to be better than currently best found swap move (BestSwap) and (3) order i has to stay filled after the swap move.

Algorithm 3 Local search for MS-Swap.

```
1:  procedure LocalSearch(solution s)
2:
3:      for each order i ∈ O do                                   ▷ O is the set of all orders
4:          ImprovementFound ← true;
5:          while ImprovementFound = true do
6:              ImprovementFound ← false;
7:              for each allocated wafer w_i in order i do
8:                  for each unallocated wafer w_j ∈ W_UA do      ▷ W_UA is the set of all unallocated wafers
9:                      if DetermineIfImprovement(BestSwap, i, w_i, w_j) = true then      ▷ See Algorithm 4
10:                         BestSwap ← SaveSwap(w_i, w_j);
11:                         ImprovementFound ← true;
12:                     end if
13:                 end for
14:             end for
15:             if ImprovementFound = true then
16:                 ImplementSwap(BestSwap);
17:             end if
18:         end while
19:     end for
```

Algorithm 4 Acceptance criterion of local search.

```
1:  procedure DetermineIfImprovement(BestSwap, order i, allocated wafer w_i, unallocated wafer
    w_j)
2:
3:      if order i is in underallocation then
4:          if Swap(w_i, w_j) decreases underallocation and Swap(w_i, w_j) is better than BestSwap then
5:              return true;
6:          end if
7:      end if
8:
9:      if order i is in overallocation then
10:         if Swap(w_i, w_j) decreases overallocation and Swap(w_i, w_j) is better than BestSwap and order
    i stays filled after Swap(w_i, w_j) then
11:             return true;
12:         end if
13:     end if
14:
15:     return false;
```

Since the objective is to minimize the total overallocation, swapping wafers among orders in many cases moves the overallocation from one order to another. Since these moves do not improve the objective they are referred to as 0-cost moves. Nevertheless, these moves can diversify the search. Capua

et al. (2018) reported that these moves have a significant contribution to the search performance. In the method proposed in Algorithm 5, 0-cost moves are considered between all possible combinations of two orders i and j. For each order combination, all possible combinations of two allocated wafers w_i and w_j are checked on feasibility, i.e. whether both orders stay filled after the swap move. If the swap between w_i and w_j is feasible, it is accepted with probability P and implemented by ImplementSwap(w_i, w_j).

Algorithm 5 0-cost moves for MS-Swap.

```
1:  procedure ZEROCOSTMOVES(solution s)
2:
3:      for each order i ∈ O do
4:          for each order j ∈ O do
5:              if i ≠ j then
6:                  for each allocated wafer w_i in order i do
7:                      for each allocated wafer w_j in order j do
8:                          if both orders stay filled after swap then
9:                              ImplementSwap(w_i, w_j) with probability P
10:                         end if
11:                     end for
12:                 end for
13:             end if
14:         end for
15:     end for
```

3.4 Gurobi Optimizer

Gurobi Optimizer is a commercial optimization solver for mathematical programming problems, such as the integer linear programming (ILP) problem introduced in Section 2. To solve ILP problems it uses a linear-programming based branch-and-bound algorithm. For a more elaborate description of this algorithm the reader is referred to (Gurobi Optimization. 2019).

4 REAL-WORLD CASE STUDY

As mentioned before, the solution method proposed in this paper is applied of Nexperia's back-end assembly and test facility in Malaysia. The product portfolio at this site is large: it contains approximately 900 unique products, using more than a 1000 different wafer types. In total, Nexperia produces more than 90 billion products annually. In this section, the steps that are required to translate the practical problem to solvable WAPs will be described.

4.1 Order Selection

In Nexperia's order planning, it is often not possible to cover all requested customers orders with the available wafer supply. In this case, the ILP presented in Section 2 would become infeasible. To prevent this, the orders first go through an order selection phase.

First the planned orders of the same finished good (FG) type are combined. Similarly, all wafers of the same type are grouped together. Then, the order-wafer relations are mapped in a bipartite graph. An illustrative example is shown in Figure 1. Orders for FG type B need wafers of type 1 and 2, while FG type D uses wafer type 2 and wafer type 3 and so forth. Clearly, two closed groups of orders and wafers are found. These disjoint groups are referred to as *matching groups*.

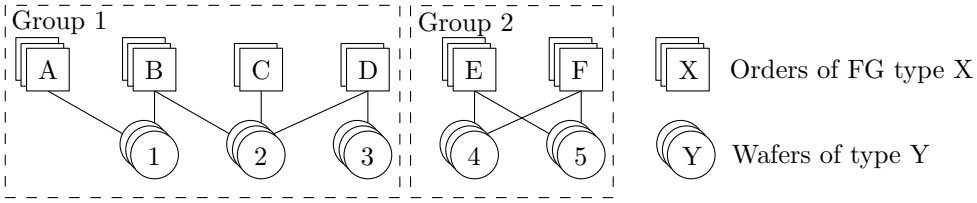

Figure 1: A bipartite graph to illustrate the order-wafer relations. This graph decomposes into two disjoint groups, so-called matching groups.

The order selection stage has many similarities with the *bin packing problem*. Recall that the bin packing problem is a dual version of the bin covering problem. For the complete formulation of the classic bin packing problem the reader is referred to Baker (1985). In this case the bins represent the available quantity of a certain wafer type and the items represent the planned orders. The only difference is that in some cases an order requires two wafer types, meaning that if selected, it fills two bins simultaneously.

The objective of the order selection stage is threefold: (1) to make the ILP problem feasible, (2) to maximize the fulfilled planned orders given the wafer supply and (3) to meet the due dates of the planned orders as much as possible. To mimic the manual order selection the *FIFO-First-Fit* policy is used. This heuristic attempts to satisfy the conflicting objectives (2) and (3). However, this policy does not suffice to ensure the first objective of ILP feasibility, an aspect that will be addressed shortly. The *FIFO-First-Fit* policy is described below:

FIFO-First-Fit for order selection:

1. For each wafer type: sum all wafer quantities and set this as the *capacity* of the corresponding bin.
2. For each planned order: translate the total FG quantity to the required wafer quantities.
3. Sort the planned orders by due date, nearest due date first.
4. Select first unallocated planned order.
5. Check if planned order fits for all corresponding wafer types. If it does not fit, go to Step 8.
6. Move order to the *selected* group.
7. Allocate order to bins of corresponding wafer types, go to Step 9.
8. Move order to the *unselected* group.
9. If there are remaining planned orders, go to Step 4.
10. Stop.

Even though the sums of the requested wafer quantity of the selected planned orders will not exceed the available wafer quantities, the ILPs can still be infeasible. This will only be noticed when the actual matching is started. If this appears to be the case, the wafer types that cause the problem to be infeasible are identified and the order with the latest due date is removed. This procedure is repeated until the ILPs become feasible.

4.2 Translate to WAPs

The matching groups require a translation step to obtain multiple WAPs. Each unique wafer type creates one ILP. For example, considering the matching groups in Figure 1, there are five different wafer types. Thus, the problem is split into five disjoint WAPs. Orders of FG type B need wafers from both wafer type 1 and 2. Therefore, the orders of FG type 2 will decompose in two separate orders B1 and B2 for WAP1 and WAP2 respectively. The size of order B1 and B2 is the sum of the requested die quantities for, respectively, wafer type 1 and wafer type 2 from all selected planned orders of FG type B. Thus, the set of orders O in WAP1 is $\{$A1, B1$\}$ and the set of wafers W consists of all wafers of type 1. Now, one of the

solution methods presented in the previous section can be applied to solve each of the WAPs separately, as will be discussed in the next section.

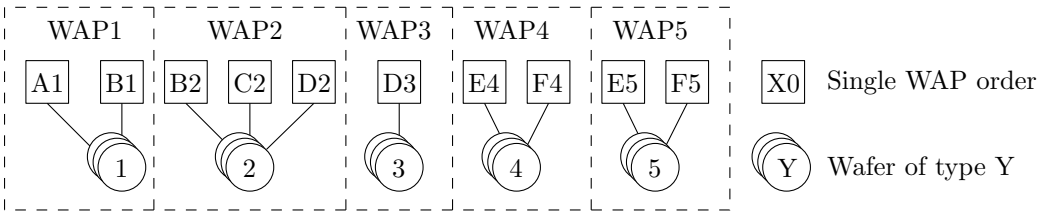

Figure 2: Schematic overview of the WAPs corresponding to the matching groups of Figure 1.

5 COMPUTATIONAL RESULTS

This section discusses the computational results of the solution methods based on several data sets of a back-end assembly and test facility of Nexperia. All algorithms are coded in C# 6.0 and all experiments are run on a computer with an Intel 2nd Generation Core i5 processor running at 3.10 GHz and 4 GB of RAM memory.

5.1 Problem Instances

Real-world production data sets of four separate weeks are obtained from Nexperia's back-end facility. One average weekly data set consists of a large number of wafers and orders, which can be decomposed into approximately 70 disjoint WAPs. The WAPs differ greatly in size, the number of orders in the largest WAP can be 15 times as much compared to the smallest one, while the number of wafers can be 50 times as large.

To make a fair comparison between the different solution methods, the planned orders in the data sets are selected such that all the solution methods are able to find feasible solutions for the same set of planned orders. For each data set, this is done by running the order selection heuristic explained in Section 4.1, with each of the solution methods in Section 3. Then, the selected planned orders of each run are compared and the smallest set of planned orders is chosen. This chosen set of planned orders is then used for all solution methods.

5.2 Parameter Settings

For the experiments in this work, the parameter settings shown in Table 1 are used. The parameters for MS-Swap are chosen based on preliminary experiments, showing that these settings yield good results within acceptable computational time for these data sets. The purpose of each of the parameters is elaborately discussed in Section 3.3. Although the MS-Swap algorithm contains randomness, these experiments also showed that for the chosen parameters the variation in the outcome is negligible.

For the Gurobi solver, all parameters are set to default, as recommended in (Gurobi Optimization. 2019), except for T_{lim} and I_{feas}. T_{lim} is chosen to be 500 seconds per disjoint group, such that the computational time is acceptable for this application. For many WAPs, Gurobi found the proven optimal solution within T_{lim}. For the WAPs where it did not, the time limit had to be increased beyond what was acceptable for the application to prove optimality. However, the current T_{lim} provided objective values which were very close to the lower bound already. Furthermore, an integrality constraint is satisfied in Gurobi if the variable's value is less than I_{feas} from the nearest integer value. This tolerance is chosen to be tight enough (1e-8), such that the value of the found objective did correspond to the same value of the calculated overallocation using the allocated wafers. Loosening this tolerance slightly reduces the runtime, but allows larger integrality violations, causing errors in the objective.

Table 1: Parameter settings for MS-Swap and Gurobi Optimizer.

Parameter settings.			
Algorithm	Parameter	Description	Value
MS-Swap	K	The maximum increase in number of wafers in each order at initialization.	2
	P	Acceptance probability of 0-cost moves.	0.5
	R	Number of iterations through the sequence of local search and 0-cost moves.	2
Gurobi	T_{lim}	Limits the total runtime (in seconds) of the solver per disjoint group.	500
	I_{feas}	Integrality restriction on a variable is considered satisfied when the variable's value is less than I_{feas} from the nearest integer value.	1e-8

5.3 Overallocation Performance

As formulated in the WAP, the objective is to minimize overallocation. Overallocation is defined as the portion of the total allocated dies which will result in overproduction. Overallocation is computed according to equation (4) where the *allocated die quantity* is the sum of dies on each allocated wafer and the *required die quantity* is the sum of requested die quantities of all the orders.

$$\text{Overallocation (OA)} = \frac{\text{Allocated die quantity} - \text{Required die quantity}}{\text{Allocated die quantity}} \cdot 100\% \tag{4}$$

Table 2 shows the performance results in terms of overallocation of all the proposed solution methods. The actual overallocation at Nexperia's back-end facility resulting from the current practice of wafer allocation is shown under Manual. The overallocation of Manual is normalized to 100% and the overallocation of all other solution methods is relative with respect to the manual allocation. For the Gurobi Optimizer, the Incumbent (Inc) and the Lower Bound (LB) are shown. The incumbent is the best feasible solution found within the set time limit. The bound LB is a lower bound on the optimal objective. The difference between the incumbent and the lower bound is known as the Gap. Thus, if the gap is zero, the found solution is proven to be optimal. Gurobi Optimizer did not prove optimality for any of the weeks, even though it is suggested by the provided decimal numbers of week 40.

Several observations can be made from these results. The first observation is that all algorithms perform substantially better than the manual current practice. This means that there are gains possible for Nexperia by implementing any of the methods of wafer allocation proposed in this work, potentially resulting in major cost savings. Secondly, the novel heuristic MS-Swap outperforms other existing heuristics such as FFD/IEG and FIFO/IEG. Lastly, for each week, the solutions provided by the MS-Swap heuristic are close to the benchmark solutions. Although the benchmarks provided by Gurobi Optimizer are not proven optimal solutions, the overall optimality gaps are very small.

The only drawback of MS-Swap compared to the existing heuristics is the computational complexity. MS-Swap is computationally more expensive than the existing heuristics. For instance, for the data set of week 41, FFD/IEG and FIFO/IEG solved the problem both in 0.9 seconds, while MS-Swap did it in 155.3 seconds. However, for the application discussed in this work, the computation time of MS-Swap is acceptable and significantly faster than current practice.

Since the time limit given to Gurobi Optimizer is acceptable in practice, it is noted that for these industrial cases both the MS-Swap and Gurobi Optimizer can be used, with a slight preference towards the latter. However, larger problem instances exist in other factories and it is unsure how either of the methods scale with the problem size.

Table 2: The overallocation performance (OA) of all algorithms: First-Fit-Decreasing with improved endgame (FFD/IEG), First-In-First-Out with improved endgame (FIFO/IEG), MS-Swap and the Gurobi Optimizer.

Overallocation performance of all algorithms.						
	Manual	FFD-IEG	FIFO-IEG	MS-Swap		Gurobi
					Inc	LB
Week	OA (%)	OA (%)	OA (%)	OA (%)	OA (%)	OA (%)
39	100	82.636	80.099	68.670	68.663	68.662
40	100	52.204	51.499	41.094	40.879	40.879
41	100	40.161	40.340	31.142	30.996	30.995
42	100	67.793	65.735	56.985	56.971	56.970
Mean	**100**	**54.187**	**53.236**	**43.842**	**43.736**	**43.736**

6 CONCLUSIONS AND FUTURE WORK

In this work, a solution method is presented, which successfully solves the problem of allocating wafers to customer orders prior to assembly at a semiconductor production facility. The objective of the solution method is minimizing the overallocation. This is important for the industrial application, because of unnecessary occupation of manufacturing equipment and overproduced products are considered excessive inventory. Some products even expire and have to be disposed. Also, less overproduction implies that more products are made which can be sold immediately, in other words, the effective capacity increases. All in all, the reduction of overproduced products has following three main advantages: (1) less inventory has to be stored, (2) less material and time is wasted on products which might be disposed and (3) more customer orders can be delivered on time.

Several existing methods and a novel heuristic are applied to real-world instances from one of Nexperia's back-end facilities. From this, it can be concluded that the proposed MS-Swap method significantly outperforms existing methods and current practice.

Both the MS-Swap heuristic as well as Gurobi Optimizer produce close-to-optimal solutions within time limits that are acceptable in practice. Further investigations are necessary to evaluate how these methods perform for larger industrial cases. Although the optimality gap achieved by Gurobi Optimizer might not be as small as here, the quality of early solutions provided by Gurobi Optimizer should be evaluated as well.

In reality more variations of the wafer allocation problem can be distinguished due to the presence of multiple die (quality) classes on the same wafer. Certain FG types may require only specific die classes. These features complicate the allocation problem, and demand for extensions of the solution methods. This is left for future research.

REFERENCES

Assmann, S., D. Johnson, D. Kleitman, and J.-T. Leung. 1984. "On a dual version of the one-dimensional bin packing problem". *Journal of Algorithms* 5(4):502–525.

Semiconductor Industry Association. 2019. "Global Billings Report History". https://www.semiconductors.org/resources/, accessed 3rd March.

Baker, B. S. 1985. "A new proof for the first-fit decreasing bin-packing algorithm". *Journal of Algorithms* 6(1):49–70.

Bischoff, E. E., and M. D. Marriott. 1990. "A comparative evaluation of heuristics for container loading". *European Journal of Operational Research* 44(2):267 – 276.

Boushell, T. G., J. W. Fowler, A. B. Keha, K. R. Knutson, and D. C. Montgomery. 2008. "Evaluation of heuristics for a class-constrained lot-to-order matching problem in semiconductor manufacturing". *International Journal of Production Research* 46(12):3143–3166.

Capua, R., Y. Frota, L. S. Ochi, and T. Vidal. 2018. "A study on exponential-size neighborhoods for the bin packing problem with conflicts". *Journal of Heuristics* 24(4):667–695.

Carlyle, M., K. Knutson, and J. Fowler. 2001. "Bin covering algorithms in the second stage of the lot to order matching problem". *Journal of the Operational Research Society* 52(11):1232–1243.

Coffman Jr., E. G., J. Csirik, G. Galambos, S. Martello, and D. Vigo. 2013. *Bin Packing Approximation Algorithms: Survey and Classification*, 455–531. New York, NY: Springer New York.

Dyckhoff, H. 1990. "A typology of cutting and packing problems". *European Journal of Operational Research* 44(2):145 – 159.

Fowler, J., K. Knutson, and M. Carlyle. 2000. "Comparison and evaluation of lot-to-order matching policies for a semiconductor assembly and test facility". *International Journal of Production Research* 38(8):1841–1853.

Frederix, F. 1996. "Planning and scheduling multi-site semiconductor production chains: a survey of needs, current practices and integration issues". In *Manufacturing Partnerships: Delivering the Promise*, 107–116.

Garey, M. R., and D. S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

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

Knutson, K., K. Kempf, J. Fowler, and M. Carlyle. 1999. "Lot-to-order matching for a semiconductor assembly and test facility". *IIE Transactions* 31(11):1103–1111.

Woeginger, G. J., and G. Zhang. 1999. "Optimal on-line algorithms for variable-sized bin covering". *Operations Research Letters* 25:47–50.

AUTHOR BIOGRAPHIES

PATRICK C. DEENEN is a PhD candidate at the Department of Industrial Engineering of the Eindhoven University of Technology and a Sr. Business Process Analyst at Nexperia Industrial Technology Engineering Centre (ITEC). He conducted this research as part of his Master's thesis combined with an internship at Nexperia. His current research interests are in the area of modeling, design and control of manufacturing systems, manufacturing optimization and supply chain. His email address is patrickdeenen@hotmail.com.

JELLE ADAN is a Sr. Business Process Analyst at Nexperia Industrial Technology Engineering Centre (ITEC) and a doctoral candidate at the Department of Industrial Engineering of the Eindhoven University of Technology, where he obtained his M.Sc. degree in Chemical Engineering in 2017. His current research interests are supply chain, manufacturing and chemical process optimization, and data mining. His email address is jelle.adan@protonmail.com.

JOEP STOKKERMANS M.Sc. is Senior Principal and R&D Innovation Manager at Nexperia Industrial Technology & Engineering Centre (ITEC). Joep graduated in Precision Engineering from Delft University of Technology in 1997. He started at Philips Semiconductors, first as mechatronic equipment designer and soon after as project manager and team leader in the development of semiconductor assembly equipment. After a short period in Asia for setting-up a service and supply chain department in Hong Kong he is responsible for the Equipment and Process Innovation and Product Creation Process at ITEC. His email address is joep.stokkermans@nexperia.com.

ALP AKCAY is an Assistant Professor in the School of Industrial Engineering at Eindhoven University of Technology. He holds a Ph.D. in Operations Management from the Tepper School of Business at Carnegie Mellon University. His research interests include statistical decision making under uncertainty, simulation design and analysis, and approximate dynamic programming with applications in manufacturing, maintenance, and supply chain management. His email address is a.e.akcay@tue.nl.

IVO J.B.F. ADAN is a Professor at the Department of Industrial Engineering of the Eindhoven University of Technology. His current research interests are in the modeling and design of manufacturing systems, warehousing systems and transportation systems, and more specifically, in the analysis of multi-dimensional Markov processes and queueing models. His email address is i.adan@tue.nl.