Computers and Chemical Engineering 42 (2012) 189–205

Contents lists available at SciVerse ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/compchemeng

# A comprehensive constraint programming approach for the rolling horizon-based scheduling of automated wet-etch stations

Juan M. Novas, Gabriela P. Henning*

*INTEC (UNL-CONICET), Güemes 3450, S3000GLN Santa Fe, Argentina*

ARTICLE INFO

*Article history:*
Received 2 October 2011
Received in revised form 4 January 2012
Accepted 4 January 2012
Available online 28 January 2012

*Keywords:*
Automated wet-etch station
Resource-constrained scheduling
Constraint-programming
Semiconductor manufacturing systems
Multiproduct batch plants

ABSTRACT

This contribution introduces a novel and comprehensive way of dealing with the automated wet-etch station (AWS) scheduling problem. It first analyzes the field and points out some very important problem aspects that have not been properly addressed yet. Then, an expressive constraint programming (CP) formulation that considers all types of AWS transfer device movements is proposed. The model accounts for both (i) loaded trips the robot makes to transfer wafer lots between consecutive baths, and (ii) empty movements that take place when the device moves itself from a bath, where it has dropped a wafer lot, to another bath where it is required to pick up a different lot. The CP approach is afterward generalized in order to implement an innovative rolling horizon methodology. The proposed method allows the continuous operation of the wet-etch station, minimizing the unproductive intervals that would otherwise appear between the scheduling periods that correspond to different wafer lot sets that are fed by the previous manufacturing step. The formulation has been extensively tested with problem instances of various dimensionalities in productivity maximization scenarios, in which makespan was chosen as the performance measure. The results have demonstrated that robot unloaded movements cannot be neglected; otherwise, they may lead to incorrect schedules. Furthermore, they have shown that AWS scheduling needs to be addressed as an on-line activity.



## 1. Introduction

The semiconductor industry is in constant expansion, essentially due to its direct connection with a diversity of high-tech global markets. It is characterized for employing expensive equipment, manufacturing products which have very short lifecycles, and for facing varying market demands. All these features entail significant costs in case of having idle capacity. In addition, facilities have to supply customers with good quality products and deliver them on time. Due to these characteristics, advanced planning methods play an important role in semiconductor factories (Pfund, Manson, & Fowler, 2006).

Semiconductor manufacturing is one of the most complex high-tech industrial processes. As it is described by Kallrath and Maindl (2006), the process that starts from raw silicon wafers comprises four stages. They are: (1) fabrication, (2) sorting (or probing), (3) assembly (or packing), and (4) testing, where the first two are grouped into the front-end process, and the last two, in the back-end process. In turn, each stage is composed of several steps, each one consisting of a large sequence of operations performed by different machines. For instance, a typical sorting sub-process comprises: direct current (DC) test, wafer burn-in and probing (wafer test), ink marking, and back grinding steps (Bang & Kim, 2011). Some authors, as Hung and Wang (1997), also consider the material fabrication, involving the production of wafers from raw silicon, as the first stage of the overall process.

Scheduling of semiconductor factories has been described by Pfund et al. (2006) as one of the most complex and challenging problems in the industrial arena. They identified some features that characterize it: a large number of processing operations, re-entrant flows, batch units, equipment failures, sequence-dependent tool setups, etc. Most of the approaches that tackle this problem are rule-based or rely on some heuristics, as the proposal of Dabbas, Chen, Fowler, and Shunk (2001), who validated the contribution of Dabbas and Fowler (1999), which combines multiple dispatching criteria into a single rule with the aim of optimizing multiple objectives simultaneously. To this end, authors used different facility models, one of which is an adaptation of a real Motorola wafer manufacturing site.

Within the semiconductor manufacturing process, fabrication stands as one of the most important and more complex sub-processes, which can take over 300 operations (Lee, Kim, Yea, & Kim, 1997). At this stage, each wafer requires essentially five major steps: cleaning, metal deposition, photolithography, etching, and ion implant. However, wafer lots do not necessarily go every time through this sequence. Very often, there are some variations: certain sub-processes are skipped, repeated, or swapped (Johri, 1993). One of the most important operations within fabrication

* Corresponding author. Tel.: +54 342 455 9175x2102; fax: +54 342 455 0944.
*E-mail address:* ghenning@intec.unl.edu.ar (G.P. Henning).

0098-1354/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compchemeng.2012.01.005

**Nomenclature**

*Sets/indices*

| | |
|---|---|
| *Jobs*/*j* | wafer lots to be scheduled |
| *Baths*/*b* | AWS baths |
| *Stages*/*st* | processing stages |
| *robot* | transfer device/robot |
| $Bath_{st}$ | bath belonging to stage *st* |
| *EtchSt* | subset of chemical/etching stages |
| *RinseSt* | subset of water/deionizing stages |
| $Stages^{j}_{it}$ | subset of processing stages required by wafer lot *j*, not executed yet at the insertion time point *it* |
| $T^{IP}_{it}$ | set of in-progress tasks at the insertion time point *it* |
| $T^{NE}_{it}$ | set of non-executed tasks at the insertion time point *it* |
| $T^{NW}_{it}$ | set of new wafer lots to be inserted in the on-going agenda, at the insertion time point *it* |

*Parameters*

| | |
|---|---|
| $pt_{jb}$ | processing time of wafer lot *j* in bath *b* |
| $rt_{jb}$ | rinsing time of wafer lot *j* in bath *b* |
| $ut_{st,st'}$ | duration of the robot empty movement between stages *st* and *st*′ |
| $lt_{st}$ | time required to transfer a wafer lot to stage *st* from the previous one |
| *it* | insertion time point |
| *tw* | width of the solution time window during which a new solution needs to be found |

*Variables*

| | |
|---|---|
| $Task_{j,st}$ | processing task of a wafer lot *j* at the stage *st*. This activity is described by means of duration, start, and end time variables |
| $Transf_{j,st}$ | transfer activity modeling the movement of the wafer lot *j* to the stage *st*, or to the output buffer, from the previous stage, or from the input buffer if *st* = 1. This activity is described in terms of duration, start, and end time variables. |

is etching. There are two types of etching processes, wet-etch and plasma etching. In particular, wet-etching processes are used to chemically remove layers from the surface of wafers during their manufacturing. These operations are carried out by modern units known as automated wet-etch stations (AWSs) or wet-etch benches. The proper operation of the wet-etch step is critical to improve the performance of the whole facility (Geiger, Kempf, & Uzsoy, 1997). Therefore, wet-etch scheduling is one of the most significant problems of a semiconductor plant. Fig. 1 shows a sketch of a semiconductor manufacturing process, highlighting the fabrication stage and the wet-etch process, on which the rest of this contribution focuses.

An automated wet-etch station includes a series of successive chemical and water/de-ionizing baths and a shared material handling system, having one or more devices/robots, in charge of moving wafer lots from one bath to another. In addition, AWSs operate under a mixed intermediate storage policy, which has to be rigorously followed in order to avoid wafer contamination due to overexposure to chemical etchants (Karimi, Tan, & Bhushan, 2004). All these features pose a tough scheduling problem in which processing tasks need to be properly synchronized with wafer transfer activities between baths, while productivity is maximized.

Since wet-etching is not an isolated process, but a step of the semiconductor fabrication, the scheduling of an AWS has to be addressed in coordination with the previous step of the process, which continuously supplies the AWS with wafer lots to be etched. Consequently, AWS scheduling is an on-line scheduling problem which needs to take into account the current status of the on-going agenda, each time a new set of wafer lots needs to be scheduled. However, most contributions that have addressed this problem, have neglected its dynamic nature, assuming that all the resources are available at the beginning of the scheduling horizon, which is not generally true in real facilities.

The AWS scheduling problem has called the attention of several research groups in the last two decades. Various mixed integer linear programming (MILP) formulations (Aguirre & Méndez, 2010; Bhushan & Karimi, 2003; Karimi et al., 2004), heuristic algorithms (Bhushan & Karimi, 2004; Geiger et al., 1997), and constraint programming approaches (Novas & Henning, 2011; Zeballos, Castro, & Méndez, 2011) have been proposed. To address the single robot AWS scheduling problem, Geiger et al. (1997) presented a two-step approach, based on a Tabu Search algorithm, which minimizes makespan. In the first step, the processing sequence of wafers on baths is obtained disregarding wafer transfer times. In the second step, these times are considered in order to coordinate robot and bath agendas. Later, Bhushan and Karimi (2003) proposed a set of continuous-time MILP formulations, which also adopt makespan as the objective function. They are associated with two main MILP models, named URM ("unlimited robot model", assuming that a robot is always available to perform any required transfer task) and ORM ("one robot model", which considers that a single robot executes all the transfer activities). First, this work compares the results obtained with the URM and ORM models in order to study the extent to which robot associated constraints are important. Then, it proposes a two-step strategy, called RCUM heuristic. In the first step of the approach, the URM model is solved to obtain the best job sequence, and, in the second step, this sequence is fixed in the ORM model, which is later solved to obtain a feasible agenda. The proposed MILP models were later reformulated by Karimi et al. (2004) to improve their computational performance. However, both papers include partial approaches to robot modeling, because they only consider its forward travelling movement, when it transfers wafer lots between consecutive baths. Hence, the approaches neglect the unloaded movements of the device.

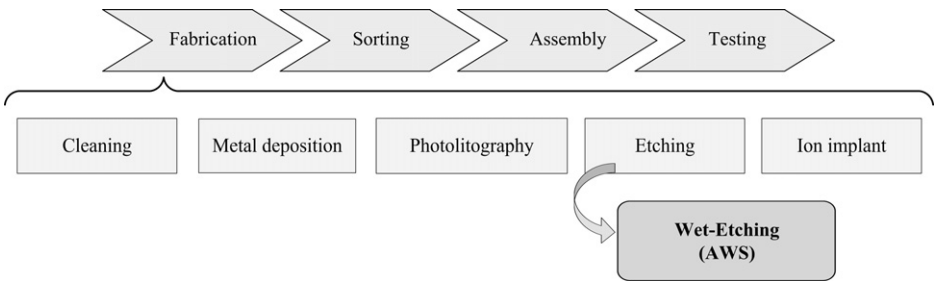


**Fig. 1.** Semiconductor manufacturing process.

More recently, the proposal of Aguirre, Méndez, and Castro (2011) also considered the URM and ORM concepts introduced by Bhushan and Karimi (2003), when developing another MILP-based solution strategy. This MILP continuous-time formulation provides the sequence and timings of the processing operations, as well as the agenda corresponding to one or more robots (depending on the AWS configuration), while minimizing the etching completion of all wafer lots. Optimal solutions were reported for small-size problems, while for medium-size examples, a decomposition method was proposed. It consists of scheduling the production activities, and then incorporating the agenda of the transfer devices by assuming that the sequence defined in the first step is fixed. By means of this heuristic, they found good quality solutions in reasonable CPU times. This approach also neglects all the robot unloaded movements.

In another recent contribution, Zeballos et al. (2011) presented a constraint programming (CP) approach, which includes both a CP model and a search strategy, in order to reduce the computational effort. They solved several test examples and concluded that their proposal improves the computational performance when compared with the same model, but using the default search strategies of the CP solver. This approach reported good quality solutions for small and medium size problems. Nevertheless, it does not take into account the movements of the empty robot.

Although most contributions have made explicit the relevance of the robot in the scheduling of AWSs (Aguirre et al., 2011; Bhushan & Karimi, 2003; Karimi et al., 2004; Zeballos et al., 2011), they only considered as transport activities the transfer of wafer lots between consecutive baths. Actually, previous works did not take into account the time employed by the robot when it moves unloaded, even when it might take longer than the transfer between adjacent baths. Despite the fact that both makespan values and job sequences obtained from models that consider unlimited and constrained robot availability might be identical or very similar in most of the situations, the actual schedules (start/end times of processing and transfer tasks) are not the same. This may lead to some serious problems, as it was pointed out by Novas and Henning (2011). In addition, when the number of wafer lots or baths is large, or when the transfer time values are big, neglecting empty robot movements may lead to processing sequences that are not optimal and unfeasible from an industrial point of view. These inconveniences, which will be shown in Section 2.1, and will be further detailed in Section 4, allow us to conclude that the constraints associated with every type of robot movement should be included in every solution proposal.

It is surprising that this empty robot movement matter has been ignored in the wet-etch domain, while it has been always taken into account in very close fields, like robotic flowshops employed in the steel and electronic industries, the scheduling of automated guided vehicle systems (AGVs) used in flexible manufacturing systems (FMSs), warehouses, container terminals, etc., or in the hoist scheduling problem (HSP) in the electroplating industry. In effect, this last category of problem has been studied for a long time and all the contributions have considered the unloaded movement of hoists. Actually, various authors have resorted to the so-called time way diagram to explicitly represent all sorts of hoist movements. Manier and Bloch (2003) have identified several classes of HSPs and have proposed a dedicated notation to make a systematic specification of problem types and the identification of studied problem instances easier. From such notation it is straightforward to recognize that the wet-etch scheduling problem is quite similar to a rather simple HSP, to which additional soaking constraints are imposed (fixed times on chemical baths and bounded times on the rinsing ones). Analogies can also be found with the robotic flowshop scheduling problem, which has also received a lot of attention in literature (Che, Chabrol, Gourgand, & Wang, 2012). Finally, unloaded device movements have also been considered in problems involving AGVs. For instance, Nishi, Hiranaka, and Grossmann (2011) have developed a bilevel decomposition algorithm for solving the simultaneous production scheduling and conflict-free routing problems for AGVs.

Another important aspect to remark is the fact that contributions devoted to the wet-etch scheduling problem assume that baths and robots are fully available at the beginning of the scheduling horizon, which is many times an unrealistic assumption given the fact that AWSs continuously receive wafer lots from the preceding manufacturing step (Fig. 1). As it is shown in Section 2.2, when developing a new schedule it is necessary to take into account the current status of the in-progress agenda in order to avoid the overlapping of the already scheduled tasks and the new ones to be incorporated in the schedule.

This contribution presents a novel CP formulation to tackle the automated wet-etch station, short-term scheduling problem. In addition to loaded movements, it addresses the empty transfers that the transfer device/robot makes every time it is requested to pick up a wafer lot from a bath located in a position which is different than the current one. Furthermore, the approach relies on the concept of scheduling as a continuous activity, in which new orders need to be scheduled considering an existing on-going agenda; i.e. both the current status of resources and the in-progress agenda, are taken into account when scheduling new wafer lots. The paper is organized as follows. In Section 2, the problem main characteristics are described, emphasizing robot-related features and the justification for a rolling horizon-based approach. Section 3 introduces the basic CP formulation and then extends it to be able to implement the rolling horizon scheduling strategy. Finally, Section 4 discusses the computational results corresponding to several test examples, and Section 5 highlights the conclusions of this work.

## 2. Problem main characteristics

An AWS is a non-intermediate storage (NIS) flowshop consisting of an input buffer, an output buffer, a sequence of alternating bath stages, and one or more transfer devices or robots. It comprises chemical and water/deionizing baths (Fig. 2). Wafer lots, each of which is composed of several wafers having the same recipe, pass through the AWS for etching and rising; i.e. each wafer lot must follow a specified sequence of baths defined in the recipe of the product. Wafers are taken from the input buffer, where they arrive from the previous semiconductor manufacturing process, and are transferred to the first stage. This stage always corresponds to a chemical bath, followed by a rinsing one that uses a water/deionizing solution; then, both bath types alternate sequentially. When a given wafer lot is finished, it is transferred to the output buffer. While chemical baths are in charge of dissolving the material deposited on the wafer in previous fabrication stages, water or deionizing baths rinse the remaining etchants before the wafer lot is placed in the subsequent chemical bath. Depending on the recipe being followed, some baths could be skipped; however, backtracking is not possible.

If a wafer lot stays in a chemical bath longer than its etching time, it is damaged. Consequently, chemical stages must stick on a zero wait (ZW) storage policy. This means that wafers have to be removed from chemical baths immediately after their prescribed processing times, and transferred to the succeeding rinsing bath in the sequence. Conversely, rinsing stages can hold wafers longer without damaging them; hence, they follow an unlimited wait (UW) storage policy; i.e. wafers can stay in water/deionizing baths longer than the given processing time. Thus, the entire process is a NIS flowshop with a mix of ZW and UW operating policies.

AWSs operate in such a way that baths can hold only one wafer lot at a time. In addition, processing times are derived from the

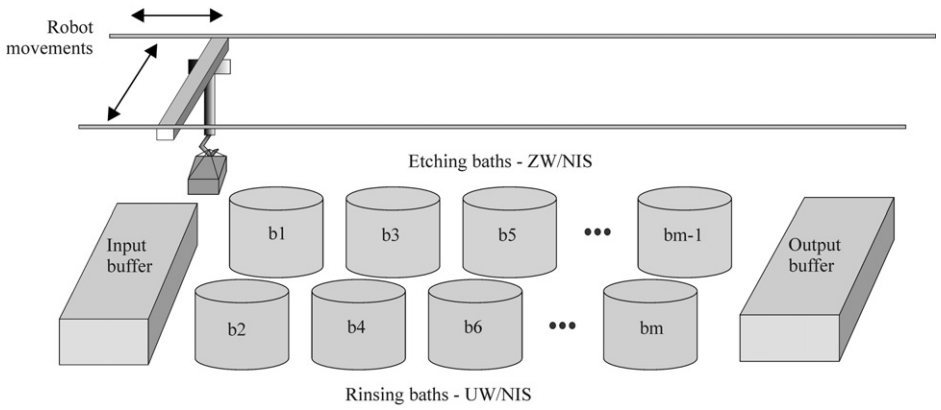


**Fig. 2.** Automated wet-etch station scheme.

wafer lot recipe and production route, i.e. they depend on both the wafer lot and the bath where it is immersed. In addition, neither the processing nor the transfer activities are pre-emptive.

In this environment, at least one robot is responsible for moving wafer lots between baths, as well as transferring them from the input buffer to the first bath, and from the last bath to the output buffer. AWS robots move one wafer lot at a time and cannot be used as storage devices.

### 2.1. Robot related features

In most AWSs the time employed by the transfer device to travel between any two baths does not depend on whether it makes a loaded or unloaded trip, but it is determined by the travel distance. Robot duties can be classified under two main types, which are described as follows:

(a) *Loaded robot movements*: They take place when the robot transfers a wafer lot between baths pertaining to two consecutive processing stages, $st'$ and $st''$, which belong to the wafer lot recipe. This type of movement also occurs (i) when the robot feeds the first stage by picking-up a wafer lot from the input buffer, and (ii) when it releases a wafer lot from the last rinsing bath, moving it to the output bath. The duration of a loaded transportation is represented by the $lt_{st''}$ parameter, in which $st''$ is the destination stage/buffer.

(b) *Unloaded robot movements*: These are empty trips executed by the transfer device, and they also take place between two baths, each one corresponding to different stages, $st$ and $st'$. These unloaded movements also occur between the output buffer and a processing stage, or between a given stage and the input buffer. The duration of an empty trip, named $ut_{st,st'}$, depends on the departure and destination baths, which belong to stages $st$ and $st'$, respectively. This type of movement is aimed at placing the robot at stage $st'$, where it is required to pick up a wafer lot that then needs to be transferred to the next processing stage, $st''$. For simplicity reasons the input and output buffers have been conceptualized as stages.

From the previous description it can be inferred that every unloaded robot trip is followed by a loaded one. Fig. 3 depicts a clarifying example, which by means of three snapshots, distinguishes different types of robot movements and their sequence. It can be observed that in order to transfer a wafer lot between two consecutive baths it is always necessary to make a prior unloaded movement, which has a duration that depends on the initial position of the robot. Due to the previous characteristics, and to the fact that most AWSs only have one robot to perform all the required movements, the transfer device becomes a critical component of the station.

As it was previously mentioned, many contributions have pointed out that the transfer device plays a critical role in the AWS scheduling; however, they have only considered the loaded robot movements between consecutive baths. Due to this fact, they may lead to sub-optimal schedules and/or to schedules that may be unfeasible to implement in practice since the robot cannot do two different things at the same time. In addition, some of the solutions reported in literature could render damaged wafers due to their overexposure to chemical etchants.

Fig. 4(a) depicts a portion of a Gantt chart that has been extracted from the literature, which shows a typical situation of an unfeasible agenda. In this figure, processing tasks are labeled with the number of the wafer lot being processed, while transfer tasks are named with the number of the lot being moved and its destination bath. The robot agenda is enlarged in Fig. 4(b) to show the detailed sequence of movements carried out by the transfer device. According to this partial agenda, at the $t_{n'}$ time point, the robot leaves the wafer lot $j9$ in bath $b8$ and picks up wafer lot $j3$ from bath $b6$. But these two things cannot occur at the same time point; i.e. the robot cannot pick up and drop different wafer lots, on distinct baths, at the same time. Indeed, the robot needs time to move itself unloaded from bath $b8$, after leaving $j9$, to bath $b6$, in order to pick up $j3$. A similar state of affairs occurs immediately after finishing the transfer of wafer lot $j3$ to bath $b7$. The solution shows that at time point $t_{n''}$, the transfer of $j3$ ends at $b7$, and when this wafer lot is being dropped there, another transfer task starts at bath $b4$, where the robot picks up wafer lot $j1$. As seen, this situation is not feasible either.

Another portion of the same schedule is depicted in Fig. 5(a) in order to show another limitation of previous contributions, concerning damaged wafers as the result of overexposure to etchants. The robot agenda corresponding to this schedule is enlarged at the right hand side of the drawing. According to this schedule, almost immediately after the movement of wafer lot $j9$ between baths $b11$ and $b12$, the transport of lot $j4$ between baths $b1$ and $b2$ takes place. Fig. 5(b)–(d) schematically represents the associated robot trips. It can be observed that after transferring $j9$ from $b11$ to $b12$ (see Fig. 5(b)), the robot has to move unloaded between baths $b12$ and $b1$ (Fig. 5(c)), in order to pick up $j4$. Finally, the device takes wafer lot $j4$ to bath $b2$ (see Fig. 5(d)).

Since the durations of the robot movements depend on the distance between baths, the time employed by the robot to move itself between baths $b12$ and $b1$ (last and first baths of the AWS) is much greater than the transfer time between any consecutive baths. Thus, this empty robot movement lasts more than the time depicted in Fig. 5 and, therefore, wafer lot $j4$ will not be picked up on time, being damaged due to the overexposure to chemical etchants.

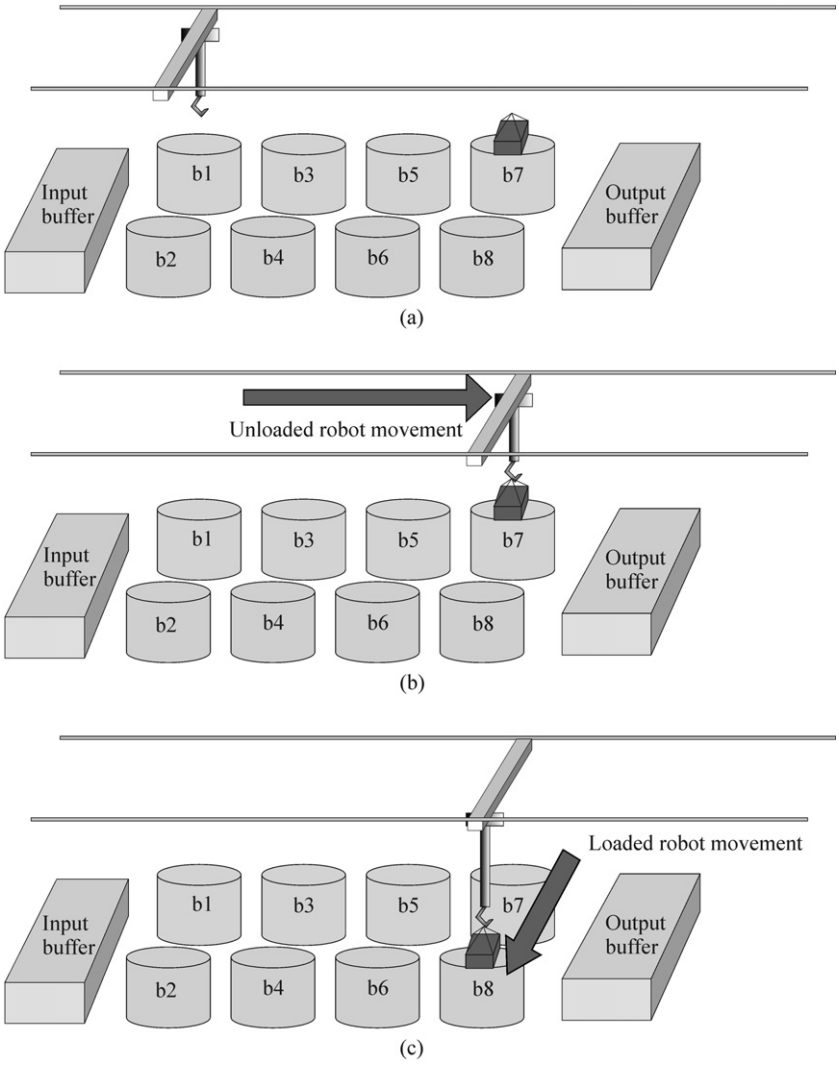


**Fig. 3.** Different types of robot movements. (a) Location of the robot ($st = b1$) when a pick up request from bath $b7$ is triggered. (b) Unloaded robot trip from $st = b1$ to $st' = b7$. (c) Loaded robot movement from $st' = b7$ to $st'' = b8$.

### 2.2. On-line scheduling issues

At the fabrication stage of semiconductor manufacturing processes, the etching step follows the photolithography one (see Fig. 1). Therefore, photolithography intermittently feeds the etching station with in-process wafer lots that need to be incorporated into the AWS agenda. Previous contributions have assumed that the AWS is fully available and ready to be used at the time point where a new set of wafer lots is included in the schedule, referred as *insertion time point*, or simply *it*. In other words, these works suppose that the whole set of wafer lots, belonging to the previous agenda, has been completely etched and is already located in the AWS output buffer. Additionally, they assume that all the resources (baths and robot/s) are available at the beginning of the scheduling horizon. However, these circumstances do not occur in industrial settings because they produce long idle times on baths, leading to a reduction in the overall station performance.

Fig. 6 shows a partial Gantt chart that schematizes the situation described above. In this diagram the insertion time point corresponds to the end of the transportation of wafer lot $j3$ (last lot to be etched according to the on-going agenda) to the output buffer $ob$. As seen, this operational policy would render long unproductive intervals on baths and cannot be implemented in practice. As a result, the diagram indicates that the insertion time point should be placed much earlier, e.g. when the processing of wafer lot $j3$ in bath $b1$ finishes. In fact, baths belonging to the first stages of the process can start executing tasks associated with the new set of wafer lots, while baths pertaining to the last stages are still carrying out activities that belong to the previous agenda. However, if scheduling is addressed in this way, it turns into an on-line type of problem in which both the in-progress agenda and the AWS status must be taken into account in order to properly handle resource availability. Indeed, to prevent the overlapping of the already scheduled activities with the new tasks to be inserted, knowledge of the AWS status at time point *it* is necessary. This status refers to the current assignments of the processing and transport activities, as well as their scheduled start and end times. In summary, the problem needs to be tackled under a rolling horizon type of approach.

Fig. 7 illustrates how disregarding the on-going robot schedule, when scheduling a new set of tasks, could lead to the overlapping of transport tasks. This diagram also shows that in order to prevent the overlapping of the processing tasks demanded by the new wafer lots (for instance $j4$), with the ones of the on-going agenda, it is necessary to take into account (i) the scheduled tasks that are not executed at *it* yet (e.g. activities demanded by wafer lot $j3$, which take place on baths $b2$, $b3$, $b4$, or by wafer lot $j2$ on bath $b4$), as well as (ii) activities that are still under execution at *it* (e.g. the one carried out in bath $b3$ on wafer lot $j2$).

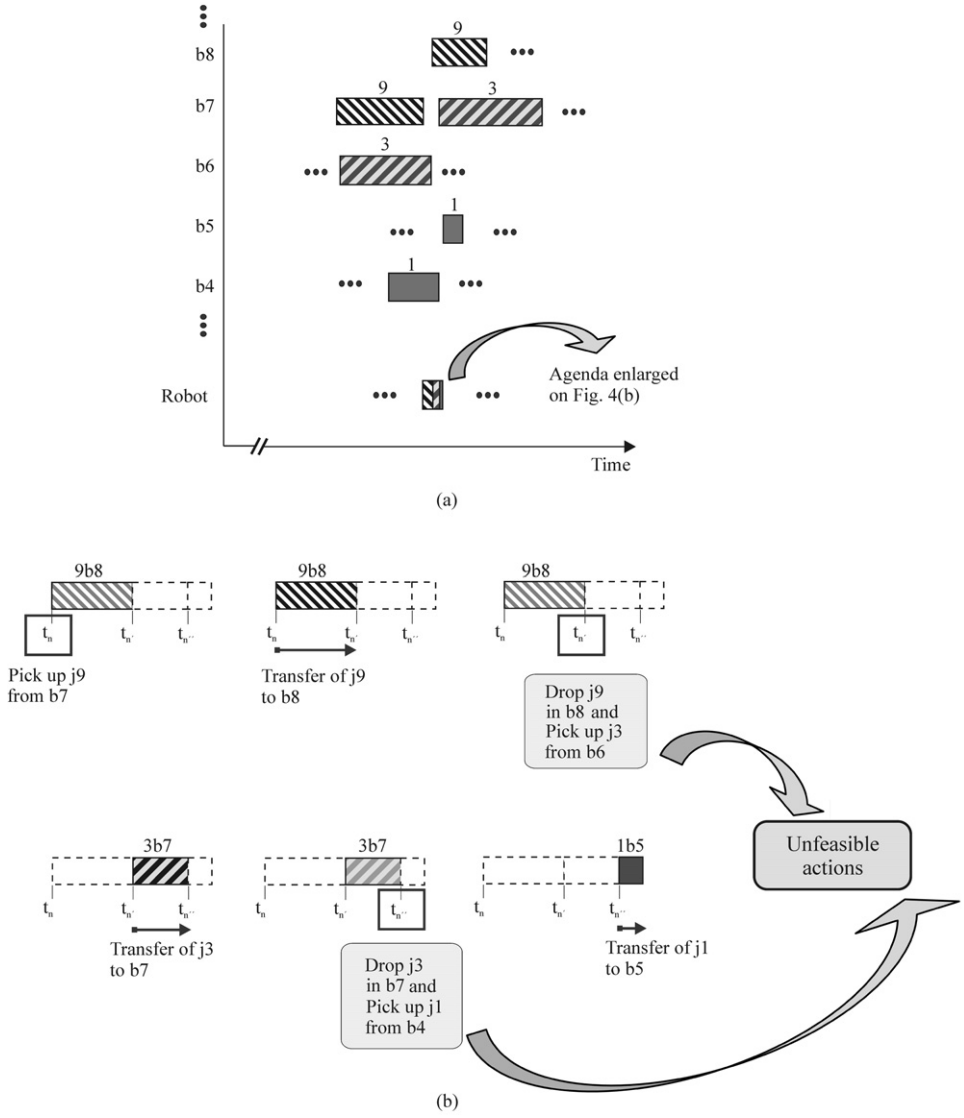


**Fig. 4.** (a) AWS partial schedule view. (b) Detailed sequence of robot movements showing unfeasible situations.

As a result, when addressing the AWS scheduling activity as an on-line type of problem it is important to consider that tasks that were programmed in the previous scheduling period are still being continued at and after the insertion time point, thus competing for resources. Therefore, it is necessary to distinguish the status of all the activities at *it*; thus, tasks belonging to the on-going agenda need to be classified into the following groups: (i) the set of already executed tasks, $T_{it}^{AE}$, which corresponds to activities belonging to the on-going agenda that have finished at or before the *it* time point, and are going to be neglected, (ii) the non-executed task set, $T_{it}^{NE}$, which gathers tasks that have not started at *it* yet, and (iii) the set of in-progress tasks, $T_{it}^{IP}$, which corresponds to the ones under execution at time point *it*. Additionally, activities demanded by the new set of wafer lots to be scheduled at insertion time point *it* are grouped into the $T_{it}^{NW}$ set. This classification is based on the reactive scheduling framework proposed by Novas and Henning (2010).

## 3. Constraint programming formulation

Constraint programming approaches have been successfully applied to a variety of scheduling problems. Formulations based on CP provide several advantages, as the capability to detect infeasibilities immediately, as well as to obtain initial feasible solutions quite fast. Moreover, optimal and suboptimal solutions can be instantiated in low CPU times. It is also important to remark that CP languages are highly declarative in nature and facilitate model development. All these features render benefits when addressing industrial problems, and due to these reasons constraint programming has been selected as the technology to address AWS scheduling.

The CP approach presented in this paper has been implemented in the OPL language, which is the underlying language of the ILOG OPL Studio environment (ILOG, 2002). It employs some specific scheduling constructs available in the ILOG Scheduler package (ILOG, 2000a). The main constructs used in this proposal are: (i) *requires*, that enforces the assignment of the renewable resources demanded by the activities, (ii) *precedes*, which imposes a proper sequence of non-overlapping tasks, and (iii) the predicate *Activity-HasSelectedResource*, which evaluates to true when a task has been assigned to a particular resource belonging to a set of alternative resources.

In this contribution, the AWS scheduling problem is tackled by means of two complementary models. First, a CP basic formulation that considers loaded and unloaded robot movements is

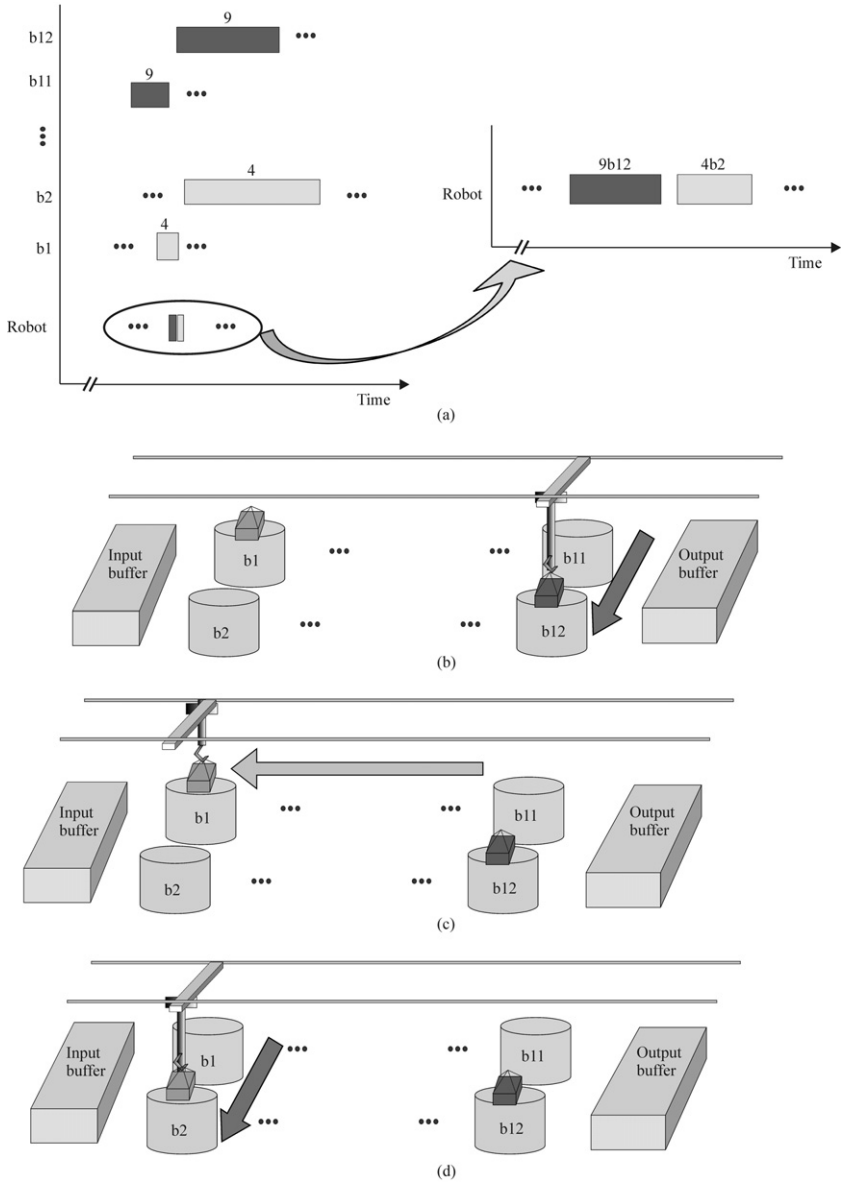


**Fig. 5.** (a) AWS partial schedule that highlights transfer tasks. (b) AWS scheme representing the transfer of $j9$ from $b11$ to $b12$. (c) Empty movement of the robot from bath $b12$ to $b1$. (d) Transfer of wafer lot $j4$ from bath $b1$ to $b2$.

introduced. It addresses the AWS scheduling problem in isolation of its context, disregarding its intrinsic continuous characteristic; i.e. it is assumed that all the resources are available at the beginning of the scheduling horizon. Afterwards, the CP model is generalized to tackle the AWS scheduling problem as an on-line activity, under a rolling horizon-based approach; i.e. the AWS schedule status at the insertion time point will be considered. Under these conditions, the objective of the constraint programming model presented in this work is: (i) to determine the sequence of wafer lots to be processed at each bath, as well as the start and end times of all the processing activities, (ii) to define a detailed robot schedule, addressing both loaded and unloaded movements of the transfer device, and (iii) to specify pick up and delivery times associated with all the robot activities.

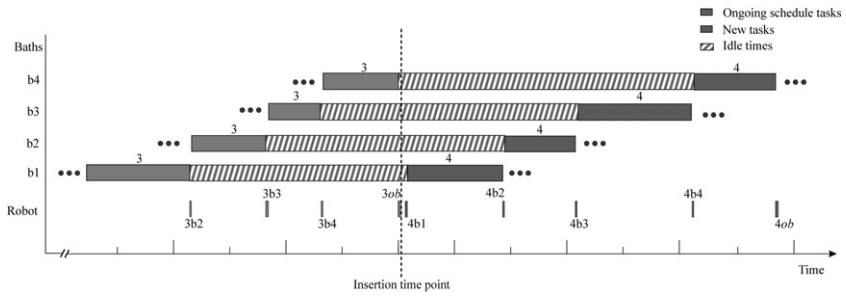


**Fig. 6.** Partial view of a Gantt chart showing the bath idle times that result when complete AWS availability is required to start a new schedule.

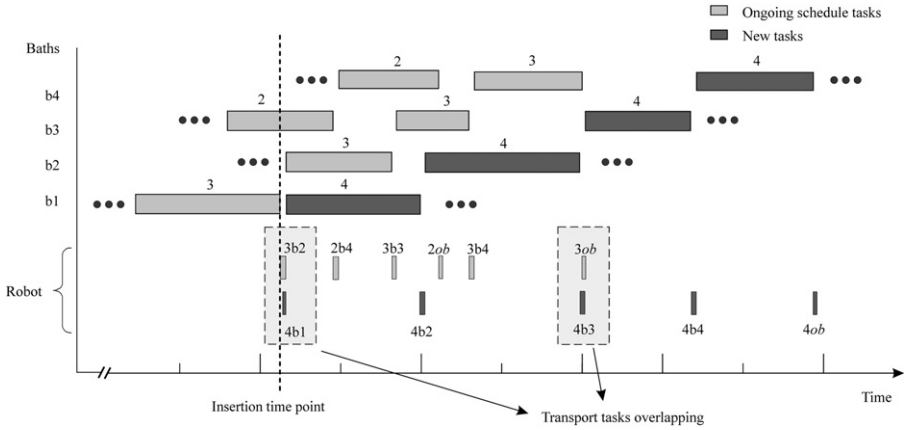


**Fig. 7.** Example showing the possible overlapping of transfer activities.

Regarding hypotheses, in this approach as well as in others reported in the literature, the following assumptions are made: (i) all the wafer lots $j$ undergo the same sequence of processing stages, (ii) no breakdowns occur during the processing of a wafer lot, (iii) each stage $st$ is composed of a single bath $b$, belonging to the set of *Baths* of the AWS, (iv) processing times of wafer lots in chemical baths, represented by $pt_{jb}$, as well as their dipping times in rinsing baths, $rt_{jb}$, are deterministic and known beforehand, (v) there is a single *robot* in charge of moving wafer lots between baths. This transport device is capable of connecting any pair of baths belonging to the AWS. Given that each stage has a single bath, the duration of the robot movements can be expressed in terms of either stages or baths, since they are equivalent. Thus, unloaded robot trips have a known duration, named $ut_{st,st'}$, which depends on both the departure and destination stages, $st$ and $st'$. On the other hand, the duration of loaded robot transfers are represented by $lt_{st''}$ and are expressed in terms of the destination stage $st''$.

The explicit modeling of stages and baths may allow, in the future, applying this approach into other domains in which more than one bath per stage might exist.

### 3.1. AWS basic scheduling model

#### 3.1.1. Assignment and precedence constraints associated with etching and rinsing activities

Constraint (1) is an assignment relation prescribing that each processing or rinsing operation required by wafer lot $j$ must be assigned to a bath belonging to the *Baths* set. This constraint works as described if, in addition, this resource has been declared as a unary one in the ILOG environment. Constraint (1) is accompanied with expression (2) that negates the *ActivityHasSelectedResource* predicate to forbid the assignment of activity $Task_{j,st}$ to any bath that does not belong to stage $st$. Constraint (3) enforces the proper sequencing of all the operations belonging to the recipe of wafer lot $j$. In this expression, $st$ and $st'$ are consecutive processing stages associated with the manufacturing of wafer lot $j$.

$$Task_{j,st}\ requires\ Baths;\quad \forall j \in Jobs,\ \forall st \in Stages \tag{1}$$

$$not\ ActivityHasSelectedResource(Task_{j,st}, Baths, b);\quad \forall j \in Jobs,\ \forall st \in Stages,\ \forall b \notin Bath_{st} \tag{2}$$

$$Task_{j,st}\ precedes\ Task_{j,st'};\quad \forall j \in Jobs,\ \forall st, st' \in Stages,\ st \neq last(Stages),\ ord(st') = ord(st) + 1 \tag{3}$$

#### 3.1.2. Wafer lot transfer activities: assignment and precedence constraints

Expression (4) is an assignment constraint prescribing that each transfer activity $Transf_{j,st}$ required by wafer lot $j$ needs the transfer device. Constraint (5) guarantees an appropriate sequencing of all the transfer operations demanded by wafer lot $j$. $Transf_{j,st}$ represents the movement of wafer lot $j$ to stage $st$ from its previous processing stage, while $Transf_{j,st'}$ models the transfer between $st$ and $st'$.

$$Transf_{j,st}\ requires\ robot;\quad \forall j \in Jobs,\ \forall st \in Stages \tag{4}$$

$$Transf_{j,st}\ precedes\ Transf_{j,st'};\quad \forall j \in Jobs,\ \forall st, st' \in Stages,\ st \neq last(Stages),\ Ord(st') = Ord(st) + 1 \tag{5}$$

#### 3.1.3. Precedence constraints relating manufacturing and transfer activities

Constraint (6) prescribes that the transfer of a wafer lot $j$ to a stage $st$ precedes the manufacturing operation (either etching or rinsing) that is carried out in such stage.

$$Transf_{j,st}precedes\ Task_{j,st};\quad \forall j \in Jobs,\ \forall st \in Stages \tag{6}$$

#### 3.1.4. Timing constraints for etching and rinsing activities

The residence time of wafer lot $j$ in bath $b$, belonging to an etching stage, should be exactly equal to the processing time $pt_{jb}$ to avoid wafer damage. As already mentioned, a NIS-ZW policy has to be followed on chemical baths to avoid the overexposure of wafers to chemicals. On the other hand, the residence time of wafer lot $j$ at bath $b$, belonging to a rinsing stage, can be greater than the rinsing time $rt_{jb}$, without any risk of $j$ being damaged. On these baths, a NIS-UW policy is followed. These two conditions are captured by constraints (7) and (8), respectively.

$$ActivityHasSelectedResource(Task_{j,st}, Baths, b) \Rightarrow Task_{j,st}.duration = pt_{jb};\quad \forall j \in Jobs,\ \forall st \in EtchSt,\ \forall b \in Bath_{st} \tag{7}$$

$$ActivityHasSelectedResource(Task_{j,st}, Baths, b) \Rightarrow Task_{j,st}.duration \geq rt_{jb};\quad \forall j \in Jobs,\ \forall st \in RinseSt,\ \forall b \in Bath_{st} \tag{8}$$

*3.1.5. Timing of transfer activities and coordination with the manufacturing ones*

The processing of a wafer lot $j$ at the bath belonging to the first stage ($st = 1$) starts as soon as its transportation ends. This occurs $lt_{st}$ time units after the movement from the input buffer ($ib$) has already started. The transfer activity is allowed to have a duration greater than the transportation time $lt_{st}$ just in case the robot remains idle after leaving the wafer lot. These conditions are captured by constraint (9).

$$\begin{aligned} &ActivityHasSelectedResource(Task_{j,st}, Bath, b) \Rightarrow \\ &Task_{j,st}.start = Transf_{j,st}.start + lt_{st} \wedge \\ &Transf_{j,st}.duration \geq lt_{st}; \\ &\forall j \in Jobs,\ st = first(Stage),\ \forall b \in Bath_{st} \end{aligned} \tag{9}$$

When a wafer lot is transferred between two consecutive stages $st$ and $st'$, its movement starts as soon as the processing in the predecessor bath finishes. In addition, the processing in the successor bath begins $lt_{st'}$ time units after the transportation has already started. Once again the transfer activity is allowed to have a duration greater than $lt_{st'}$ just in case the robot remains idle after dropping the wafer lot. These conditions are captured by constraint (10).

$$\begin{aligned} &ActivityHasSelectedResource(Task_{j,st}, Baths, b) \wedge \\ &ActivityHasSelectedResource(Task_{j,st'}, Bath, bb) \Rightarrow \\ &Transf_{j,st'}.start = Task_{j,st}.end \wedge \\ &Task_{j,st'}.start = Transf_{j,st'}.start + lt_{st'} \wedge \\ &Transf_{j,st'}.duration \geq lt_{st'}; \\ &\forall j \in Jobs,\ \forall st, st' \in Stages,\ st' \neq last(Stages), \\ &Ord(st') = Ord(st) + 1,\ \forall b \in Bath_{st},\ \forall bb \in Bath_{st'} \end{aligned} \tag{10}$$

Similarly, when the processing in the last water/de-ionizing stage ends, the wafer lot can start its movement towards the output buffer ($ob$), which for simplicity reasons has been modeled as a final stage. This transportation activity is also allowed to have a duration greater than $lt_{st'}$ just in case the robot remains idle after leaving the wafer lot into the output buffer. These conditions are represented through constraint (11).

$$\begin{aligned} &ActivityHasSelectedResource(Task_{j,st}, Bath, b) \Rightarrow \\ &Task_{j,st}.end \leq Transf_{j,st'}.start \wedge \\ &Transf_{j,st'}.duration \geq lt_{st'}; \\ &\forall j \in Jobs,\ \forall st, st' \in Stages,\ st' = last(Stages), \\ &\forall b \in Bath_{st},\ Ord(st') = Ord(st) + 1 \end{aligned} \tag{11}$$

Finally, constraint (12) captures the synchronization condition prescribing that the transfer of a wafer lot $j$ from bath $b$, belonging to stage $st$, to the following stage $st'$, has to take place before another wafer lot $j'$ arrives to such bath. This condition is enforced because baths can process only one wafer lot at a time. Therefore, a submerged wafer lot must be taken out of the bath before another wafer lot is dropped into the bath.

$$\begin{aligned} &ActivityHasSelectedResource(Task_{j,st}, Bath, b) \wedge \\ &ActivityHasSelectedResource(Task_{j',st}, Bath, b) \Rightarrow \\ &Task_{j',st}.start \geq Task_{j,st}.end + lt_{st'} + lt_{st} \vee \\ &Task_{j,st}.start \geq Task_{j',st}.end + lt_{st} + lt_{st'}; \\ &\forall j, j' \in Jobs,\ j \neq j',\ \forall st, st' \in Stages, \\ &Ord(st') = Ord(st) + 1,\ st \neq last(Stages),\ \forall b \in Bath_{st} \end{aligned} \tag{12}$$

*3.1.6. Movement of the unloaded transfer device/robot between baths*

Unloaded movements of the transfer device occur between any two baths, $b$ and $b'$, each one corresponding to a different stage, $st$ and $st'$. Since it is assumed that each stage has a single bath, robot movements can be expressed either in terms of stages or baths. Empty robot trips are intended to locate the robot at stage $st'$, each time it is required to go there to pick up a wafer lot to be taken to the next processing stage $st''$. The time employed by the robot to make an empty move depends on both the departure and destination stages, $st$ and $st'$, respectively. Stage $st$ stands for the current position of the robot, and $st'$ for the destination, which is also the departure spot for the next loaded trip to be made by the robot. Thus, empty trips can be seen as preparation activities carried out by the robot before conveying wafer lots. Consequently, they are represented as stage-dependent changeovers associated with the transfer device.

To associate transition times with the unloaded movements of the robot, every stage $st$, is linked to a $state_{st}$, as indicated in (13). In addition, transfer tasks and devices need to be declared in a special way by using the *state* and the *TransitionType* ILOG Scheduler constructs. The declaration shown in (14) associates any transfer activity required by wafer lot $j$ with the state of stage $st$, which is its destination. Furthermore, transition times need to be declared. Expression (15) defines them as a matrix having as many rows and columns as the number of stages of the process plus one. Additionally, the robot is declared to include transition times, as shown in expression (16).

$$Stages\ state[Stages] \tag{13}$$

$$Transf_{j,st}\ TransitionType\ state_{st};\quad \forall j \in Jobs,\ \forall st \in Stages \tag{14}$$

$$TransitionTime[Stages, Stages] \tag{15}$$

$$UnaryResource\ robot(TransitionTime) \tag{16}$$

*3.1.7. Objective function*

In this contribution, makespan is the performance measure chosen to be minimized. The aim is to find a schedule with the minimum total time demanded to complete all the wafer lots included in the problem been solved. Since the minimization of makespan ($Mk$) is pursued, expression (17) has to be incorporated into the model. In addition, constraint (18) has to be included in the formulation. It enforces all transfer activities concerning the transport of wafer lots to the output buffer to end at most at the $Mk$ value.

$$minimize\quad Mk \tag{17}$$

$$Tranf_{j,st}.\ \ end\ \ precedes\ \ Mk;\quad \forall j \in Jobs,\ st = last(Stages) \tag{18}$$

*3.2. Rolling horizon-based scheduling approach*

The approach presented in this section extends the basic CP model introduced in the previous one. Therefore, when solving the AWS scheduling problem under a rolling horizon-based policy, the

constraints to be considered are those described in the basic formulation, as well as the ones to be described in the remaining of this section. The proposed methodology ensures the proper matching of the on-going AWS agenda and the new set of wafer lots to be scheduled, while preventing (i) the generation of unproductive intervals, like the ones shown in Fig. 6 and (ii) the overlapping of different processing and/or transfer tasks that compete for the same resources. Moreover, the approach attempts to avoid major changes in the current agenda. To handle this last issue, two alternative scenarios are proposed: (i) freezing up all the tasks that are already scheduled, but not executed yet, (ii) giving these activities some flexibility.

Before elaborating on these two scenarios and their associated constraints, some problem elements need to be specified. Given an in-progress AWS agenda, referred in this section as the old agenda, and a new set of wafer lots to be scheduled, it is necessary to identify:

(a) The insertion time point, *it*. This is the earliest time point at which the set of new wafer lots can be inserted in the in-progress agenda, while avoiding any change in the current lot sequence. The value of *it* is generally adopted as the completion time of the last wafer lot in sequence of the first bath. However, an earlier value can be adopted under certain circumstances. This parameter can be interpreted as the beginning of the time horizon corresponding to the new agenda.
(b) The time-window during which the new scheduling problem is to be solved. With respect to this problem element, it will be assumed that during the time interval in which the new schedule is being generated, the AWS status will not vary. Otherwise, if changes occur before the new schedule is implemented, the agenda will no longer be feasible. To avoid this situation, it is defined a time window of width *tw*, during which the solution of the new scheduling problem needs to be found, in order to be implemented at time $it + tw$. A conservative value of *tw* could be established as the difference between the earliest finishing time among the completion times of the etching tasks that are actually being executed at the insertion time point (i.e. in-progress etching tasks) and the *it* value itself. In case this *tw* value were too large, a smaller one can be adopted to avoid a loss of time. However, the lower the value of *tw*, the smaller the available time to reach a good quality solution.
(c) The set of tasks that is actually involved in the new scheduling problem. Regarding this issue, activities belonging to the old agenda are first classified into the $T_{it}^{AE}$, $T_{it}^{IP}$ and $T_{it}^{NE}$ sets. Then, the new schedule is built by taking into account the elements of $T_{it}^{IP}$ and $T_{it}^{NE}$, plus the activities associated with the new set of wafer lots, which are included in $T_{it}^{NW}$.

Given these problem elements, the rolling horizon approach proposed in this contribution can be described as follows:

1. Specify the value of *it*, the insertion time point.
2. Estimate *tw*, the width of the solution time window.
3. Classify tasks belonging to the current agenda in sets $T_{it}^{AE}$, $T_{it}^{IP}$ and $T_{it}^{NE}$, according to their status at the *it* time point.
4. Build set $T_{it}^{NW}$. It includes all the etching and rinsing tasks demanded by the new set of wafer lots.
5. Develop the CP model by taking into account tasks in $T_{it}^{IP}$, $T_{it}^{NE}$, and $T_{it}^{NW}$. If tasks belonging to the old schedule that are not already executed at *it* (i.e. activities in $T_{it}^{IP}$, $T_{it}^{NE}$) are considered to be frozen, constraints (19) and (24) need to be added to the basic CP model; otherwise, if some flexibility is given to them, constraints (20)–(24) are added to the CP model.
6. Solve the CP model by imposing a limit of *tw* time units to the allowed CPU time. The best solution achieved within this time bound can be implemented on the shop-floor at time point $it + tw$.

### 3.2.1. No flexibility scenario

In order to avoid disrupting the on-going agenda when inserting tasks belonging to the $T_{it}^{NW}$ set, activities in sets $T_{it}^{IP}$ and $T_{it}^{NE}$ are enforced to maintain their current start times at the assigned baths. This condition is captured by expression (19).

$$\begin{aligned}
&ActivityHasSelectedResource(Task_{j,st}, Baths, b) \wedge \\
&notActivityHasSelectedResource(Task_{j,st}, Baths, b') \Rightarrow \\
&Task_{j,st}.start = Task_{j,st}.plannedStart \wedge \\
&Task_{j,st}.duration = Task_{j,st}.plannedDuration; \\
&\forall Task_{j,st} \in \{T_{it}^{IP} \cup T_{it}^{NE}\},\ \forall st \in Stages,\ \forall b, b' \in Bath_{st}, \\
&b = Task_{j,st}.assignedBath,\ b \neq b'
\end{aligned} \tag{19}$$

Expression (19) indicates that the start times and durations of the activities that belong to the $T_{it}^{IP}$ and $T_{it}^{NE}$ sets, which are also part of the new schedule, are going to be exactly the same to the ones they originally had in the previous agenda, represented by the $Task_{j,st}.plannedStart$ and $Task_{j,st}.plannedDuration$ parameters.

### 3.2.2. Limited flexibility scenario

By allowing tasks in sets $T_{it}^{IP}$ and $T_{it}^{NE}$ to make minor modifications on their current agenda, it might be possible to obtain better quality solutions, at the expense of a slightly higher computational effort. Tasks in set $T_{it}^{IP}$, which are actually being executed, are not allowed to change their current start time. However, in-progress tasks associated with rinsing operations can stay longer than the prescribed time in their water baths. This will take place in those cases the extra time renders some flexibility that can be employed to make a proper coupling of both, the old and new schedules. This situation is captured by constraint (20), which allows in-progress rinsing tasks to modify their planned duration. As seen, they can increase their duration by a $\beta$ factor, but without exceeding the limit imposed by the solution time window. By definition $\beta$ has to be equal to or greater than 1. In the examples solved to test the idea, $\beta$ was assigned a maximum value of 2 ($1 \leq \beta \leq 2$), i.e. rising operations were allowed to have durations that at most duplicate the nominal rinsing times.

$$\begin{aligned}
&ActivityHasSelectedResource(Task_{j,st}, Baths, b) \wedge \\
&not\ ActivityHasSelectedResource(Task_{j,st}, Baths, b') \Rightarrow \\
&Task_{j,st}.start = Task_{j,st}.plannedStart \wedge \\
&Task_{j,st}.duration \geq rt_{j,b} \wedge Task_{j,st}.end \geq it + tw \wedge \\
&Task_{j,st}.duration \leq \beta \times rt_{j,b}; \\
&\forall Task_{j,st} \in T_{it}^{IP},\ \forall st \in RinseSt,\ \forall b, b' \in Bath_{st}, \\
&b = Task_{j,st}.assignedBath,\ b \neq b'
\end{aligned} \tag{20}$$

On the other hand, the duration of those etching tasks that belong to the $T_{it}^{IP}$ set has to be exactly equal to the prescribed processing time, $pt_{jb}$, in order to avoid overexposure to chemicals. This condition is captured by expression (21), where $Task_{j,st}.plannedStart$

and $Task_{j,st}.plannedDuration$ are the start times and durations that these activities had in the old schedule.

$$\begin{aligned}&ActivityHasSelectedResource(Task_{j,st}, Baths, b) \wedge\\ &not\ ActivityHasSelectedResource(Task_{j,st}, Baths, b') \Rightarrow\\ &Task_{j,st}.start = Task_{j,st}.plannedStart \wedge\\ &Task_{j,st}.duration = Task_{j,st}.plannedDuration;\\ &\forall Task_{j,st} \in T_{it}^{IP},\ \forall st \in EtchSt,\ \forall b, b' \in Bath_{st},\\ &b = Task_{j,st}.assignedBath,\ b \neq b'\end{aligned} \tag{21}$$

Regarding tasks belonging to the old agenda that are not so far executed at the insertion time point, i.e. activities in the $T_{it}^{NE}$ set, they are allowed to slightly change their start times. In certain cases this is absolutely necessary because their preceding rinsing tasks might have a longer duration, as prescribed by (20). In addition, rinsing tasks in $T_{it}^{NE}$ are also permitted to increase their associated durations. These conditions are represented by constraints (22) and (23). The value of $\delta$ is chosen in such a way that the wafer lot sequence of the old schedule is not modified when developing the one of the new agenda.

$$\begin{aligned}&ActivityHasSelectedResource(Task_{j,st}, Baths, b) \wedge\\ &not\ ActivityHasSelectedResource(Task_{j,st}, Baths, b') \Rightarrow\\ &Task_{j,st}.start \geq Task_{j,st}.plannedStart \wedge\\ &Task_{j,st}.start \leq Task_{j,st}.plannedStart + \delta \wedge\\ &Task_{j,st}.duration = Task_{j,st}.plannedDuration;\\ &\forall Task_{j,st} \in T_{it}^{NE},\ \forall st \in EtchSt,\ \forall b, b' \in Bath_{st},\\ &b = Task_{j,st}.assignedBath,\ b \neq b'\end{aligned} \tag{22}$$

$$\begin{aligned}&ActivityHasSelectedResource(Task_{j,st}, Baths, b) \wedge\\ &not\ ActivityHasSelectedResource(Task_{j,st}, Baths, b') \Rightarrow\\ &Task_{j,st}.start \geq Task_{j,st}.plannedStart \wedge\\ &Task_{j,st}.start \leq Task_{j,st}.plannedStart + \delta \wedge\\ &Task_{j,st}.duration \geq rt_{j,b} \wedge Task_{j,st}.duration \leq \beta \times rt_{j,b};\\ &\forall Task_{j,st} \in T_{it}^{NE},\ \forall st \in RinseSt,\ \forall b, b' \in Bath_{st},\\ &b = Task_{j,st}.assignedBath,\ b \neq b'\end{aligned} \tag{23}$$

### 3.2.3. Scheduling of activities in the $T_{it}^{NW}$ set

The scheduling of the activities required by the new wafer lots, which are included in the $T_{it}^{NW}$ set, is modeled by the constraints that comprise the CP basic model. In addition, it is necessary to ensure that tasks in the $T_{it}^{NW}$ set are going to be scheduled after the insertion time point plus the solution time limit, which is enforced by the expression (24).

$$Task_{j,st}.start \geq it + tw; \quad \forall Task_{j,st} \in T_{it}^{NW} \tag{24}$$

## 4. Computational results and discussion

The proposed CP model has been tested with several case studies of various sizes. Examples with the following dimensionalities have been solved: $Baths \times Jobs = 4 \times 5$; $4 \times 7$; $4 \times 9$; $4 \times 11$; $4 \times 13$; $6 \times 5$; $6 \times 7$; $6 \times 9$; $6 \times 11$; $6 \times 13$; $8 \times 5$; $8 \times 7$; $8 \times 9$; $10 \times 5$; $10 \times 7$; $10 \times 9$; $12 \times 5$; $12 \times 7$. Processing times have been taken from Bhushan and Karimi (2004), who generated the test data following the methodology earlier presented by Geiger et al. (1997). It consists on obtaining the processing times with a uniform distribution having a mean of 6.67 units, using 0.549 as the coefficient of variation for the residence times in chemical baths, and 0.042 as the coefficient of variation for residence times at the rinsing baths.

### 4.1. AWS scheduling. Robot related issues

In order to allow a proper comparison of the proposed approach with previous contributions, the examples discussed in this section will neglect the fact that AWSs operate in industrial environments under a rolling horizon policy. In other words, it will be assumed that all the resources are available and the tasks pertaining to the previous agenda are already finished.

All the case studies consider a single robot as the available transfer device. In addition, two scenarios have been defined in order to evaluate the CP approach:

(a) The ERMN ("empty robot movements neglected") scenario, which ignores the movements of the unloaded transfer device, as previous contributions did. This scenario is based on the same assumptions that have been made by Bhushan and Karimi (2003), Aguirre et al. (2011) and Zeballos et al. (2011). These proposals made the simplifications that were already discussed in Section 2.
(b) The ERMC ("empty robot movements considered") scenario, which takes into account the unloaded movements of the robot that have been presented in Section 2.1.

Table 1 shows the distance dependent travel times associated with robot void movements between any two stages $st$ and $st'$, represented by the $ut_{st,st'}$ parameter. These empty transfer times have been calculated from the data already available for loaded movements between consecutive stages, which are presented in Table 2. For instance, the time to move from stage $st1$ to stage $st3$, $ut_{st1,st3}$,

**Table 1**
Empty transfer times associated with unloaded robot movements between any two stages $st$ and $st'$.

| Origin stage $st$ | Destination stage $st'$ | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ib* | *st1* | *st2* | *st3* | *st4* | *st5* | *st6* | *st7* | *st8* | *st9* | *st10* | *st11* | *st12* |
| *st1* | 0.10 | 0.00 | 0.20 | 0.35 | 0.53 | 0.78 | 0.93 | 1.05 | 1.18 | 1.32 | 1.53 | 1.71 | 1.88 |
| *st2* | 0.30 | 0.20 | 0.00 | 0.15 | 0.33 | 0.58 | 0.73 | 0.85 | 0.98 | 1.12 | 1.33 | 1.51 | 1.68 |
| *st3* | 0.45 | 0.35 | 0.15 | 0.00 | 0.18 | 0.43 | 0.58 | 0.70 | 0.83 | 0.97 | 1.18 | 1.36 | 1.53 |
| *st4* | 0.63 | 0.53 | 0.33 | 0.18 | 0.00 | 0.25 | 0.40 | 0.52 | 0.65 | 0.79 | 1.00 | 1.18 | 1.35 |
| *st5* | 0.88 | 0.78 | 0.58 | 0.43 | 0.25 | 0.00 | 0.15 | 0.27 | 0.40 | 0.54 | 0.75 | 0.93 | 1.10 |
| *st6* | 1.03 | 0.93 | 0.73 | 0.58 | 0.40 | 0.15 | 0.00 | 0.12 | 0.25 | 0.39 | 0.60 | 0.78 | 0.95 |
| *st7* | 1.15 | 1.05 | 0.85 | 0.70 | 0.52 | 0.27 | 0.12 | 0.00 | 0.13 | 0.27 | 0.48 | 0.66 | 0.83 |
| *st8* | 1.28 | 1.18 | 0.98 | 0.83 | 0.65 | 0.40 | 0.25 | 0.13 | 0.00 | 0.14 | 0.35 | 0.53 | 0.70 |
| *st9* | 1.42 | 1.32 | 1.12 | 0.97 | 0.79 | 0.54 | 0.39 | 0.27 | 0.14 | 0.00 | 0.21 | 0.39 | 0.56 |
| *st10* | 1.63 | 1.53 | 1.33 | 1.18 | 1.00 | 0.75 | 0.60 | 0.48 | 0.35 | 0.21 | 0.00 | 0.18 | 0.35 |
| *st11* | 1.81 | 1.71 | 1.51 | 1.36 | 1.18 | 0.93 | 0.78 | 0.66 | 0.54 | 0.39 | 0.18 | 0.00 | 0.17 |
| *st12* | 1.98 | 1.88 | 1.68 | 1.53 | 1.35 | 1.10 | 0.95 | 0.83 | 0.70 | 0.56 | 0.35 | 0.17 | 0.00 |
| *ob* | 2.14 | 2.04 | 1.84 | 1.69 | 1.51 | 1.26 | 1.11 | 0.99 | 0.86 | 0.72 | 0.51 | 0.33 | 0.16 |

**Table 2**
Loaded robot transfer times between $st'$ and $st''$, where $ord(st'') = ord(st') + 1$, including the output buffer, and from *ib* to *st*1.

| *st*1 | *st*2 | *st*3 | *st*4 | *st*5 | *st*6 | *st*7 | *st*8 | *st*9 | *st*10 | *st*11 | *st*12 | *ob* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.10 | 0.20 | 0.15 | 0.175 | 0.25 | 0.15 | 0.12 | 0.13 | 0.14 | 0.21 | 0.18 | 0.17 | 0.16 |

**Table 3**
Computational results for various problem instances, considering both the ERMN and ERMC scenarios.

| *Baths* × *Jobs* | Constraints | Variables | ERMN scenario Optimal/best solution in 1000 s | | ERMC scenario Optimal/best solution in 1000 s | |
|---|---|---|---|---|---|---|
| | | | Makespan | CPU time[a] | Makespan | CPU time[a] |
| 4 × 5 | 345 | 151 | 61.75 | <1 | 63.17 | <1 |
| 4 × 7 | 539 | 211 | 75.93 | 2.5 | 78.13 | 5.2 |
| 4 × 9 | 765 | 271 | 90.43 | 203.7 | 93.29 | 985.5 |
| 4 × 11 | 1023 | 331 | 110.06[b] | 431.2 | 114.05[b] | 240.7 |
| 4 × 13 | 1313 | 391 | 137.44[b] | 619.4 | 143.41[b] | 118.3 |
| 6 × 5 | 565 | 211 | 72.29 | <1 | 73.57 | <1 |
| 6 × 7 | 875 | 295 | 90.71 | 7.3 | 93.35 | 27.9 |
| 6 × 9 | 1233 | 379 | 104.53 | 987.3 | 115.35[b] | 9.9 |
| 6 × 11 | 1639 | 463 | 128.38[b] | 9.4 | 131.93[b] | 382.7 |
| 6 × 13 | 2093 | 547 | 149.83[b] | 15.2 | 154.73[b] | 82.7 |
| 8 × 5 | 825 | 271 | 89.92 | <1 | 92.20 | <1 |
| 8 × 7 | 1267 | 379 | 106.41 | 46.3 | 109.24 | 123.1 |
| 10 × 7 | 1715 | 463 | 119.46 | 40.3 | 123.19 | 610.2 |
| 10 × 9 | 2385 | 595 | 139.93[b] | 366.5 | 148.55[b] | 790.3 |
| 12 × 5 | 1465 | 391 | 127.04 | <1 | 128.30 | 30.5 |
| 12 × 7 | 2219 | 574 | 140.57 | 672.2 | 144.08[b] | 731.2 |

[a] Time required to reach optimal solutions or to instantiate suboptimal ones.
[b] Best suboptimal solution instantiated within 1000 s.

is equal to $lt_{st2} + lt_{st3}$, the loaded transfer times to reach stages *st*2 and *st*3, respectively.

Table 2 depicts the transfer times of the loaded robot movements, $lt_{st''}$, which always correspond to the elapsed time to reach the bath belonging to stage $st''$, from its previous stage $st'$. These loaded transfer times have been taken from Bhushan and Karimi (2004).

The ILOG OPL Studio environment (ILOG, 2002), with the ILOG Scheduler package (ILOG, 2000a) and the ILOG Solver (ILOG, 2000b), was used to solve all the problem instances. Regardless the addressed scenario, in all the case studies, the default search strategy embedded in ILOG OPL Studio (Depth First Search, DFS), has been adopted. Examples have been solved with a computer having an Intel Pentium Dual Core 3.40 GHz processor with 2.00 GB of RAM.

Computational results for the different problem instances are shown in Table 3. The proposed CP approach has rendered optimal solutions for almost all the small and medium size problems being tackled. For both scenarios the solutions have been reached in low or reasonable CPU times. When the transfer times of the empty robot are included in the model (ERMC scenario), the approach yields solutions exhibiting greater values of makespan, thus demonstrating the importance of this problem element. Although the differences in makespan nominal values between the ERMN and ERMC solutions are not so important, the agendas of baths and robot are quite different, depending on the scenario.

Table 3 shows that optimal solutions were reached in very low CPU times for small and most medium size examples, as for the 4 × 5, 4 × 7, 4 × 9, 6 × 5, 6 × 7, 8 × 5, 8 × 7, 10 × 7, 12 × 5 case studies, either under the ERMN or the ERMC scenario. On the other hand, for bigger size examples, such as the 6 × 9 and 12 × 7 problem instances, optimal solutions have been obtained with the ERMN scenario, but suboptimal solutions under the ERMC one. In order to assess the quality of these suboptimal solutions, the 1000 s upper bound on the CPU time was removed for the 6 × 9 and 12 × 7 problem instances and these problems were solved to optimality. In the first case, a makespan value of 108.03 (108 min, 1.8 s) was obtained in 4734 s, whereas for the 12 × 7 example, a makespan of 142.72 (142 min, 43 s) was reached in 2720 s. These results indicate that, in the worst case, a value of makespan that is 6.7% above the optimal one is attained.

As it was expected, the consideration of transition times (see expressions (13)–(16)) in the ERMC model, increases the computational effort with respect to the ERMN scenario. In addition, the differences in CPU times are more significant as the problem size increases. On the other hand, the first feasible solutions corresponding to the ERMC scenario (see Table 4) were attained in very low CPU times in most cases. These results show an important characteristic of this approach: the capability of reaching feasible solutions almost immediately for most problem instances.

Another aspect that is very important to remark is the fact that by taking into account the unloaded robot trips, the structure of the AWS schedule changes in many problem instances, i.e. the main scheduling decision, which is the wafer lot processing order, is heavily affected when these trips are included in the model. Indeed, empty robot trips impose new constraints that may lead, in many cases, to a new schedule structure. This situation is illustrated in Table 5, which shows the dissimilar wafer lot sequences that

**Table 4**
First solutions found for some medium and big size examples under the ERMC scenario.

| *Baths* × *Jobs* | ERMC scenario First solution | |
|---|---|---|
| | Makespan | CPU time |
| 4 × 9 | 97.73 | <1 |
| 4 × 11 | 116.99 | <1 |
| 4 × 13 | 146.57 | <1 |
| 6 × 9 | 118.63 | <1 |
| 6 × 11 | 139.29 | 2.0 |
| 8 × 7 | 114.88 | 1.5 |
| 10 × 7 | 135.61 | 3.0 |
| 10 × 9 | 149.09 | 1.5 |
| 12 × 5 | 133.94 | <1 |
| 12 × 7 | 149.82 | 4.0 |

**Table 5**
Different job sequences reached under the ERMN and ERMC scenarios.

| *Baths* × *Jobs* | Optimal job sequence | |
|---|---|---|
| | ERMN scenario | ERMC scenario |
| 4 × 9 | 9–5–4–2–8–1–7–3–6 | 9–5–2–4–1–8–7–3–6 |
| 6 × 7 | 4–2–7–6–5–1–3 | 6–7–5–2–4–1–3 |
| 6 × 9 | 9–6–8–4–2–7–5–1–3 | 9–6–8–7–2–5–4–1–3 |
| 10 × 7 | 6–3–4–2–7–1–5 | 6–3–2–1–4–7–5 |
| 12 × 7 | 6–3–2–5–1–4–7 | 4–2–5–3–1–7–6 |

were obtained for the 4 × 9, 6 × 7, 6 × 9, 10 × 7 and 12 × 7 problem instances. In fact, in 42% of the examples in which optimal solutions were found for both the ERMN and ERMC scenarios, the corresponding schedule structures were different. It can be noticed that the cases which exhibit this behavior are the ones having a large number of baths and/or a big number of wafer lots to be processed.

Another feature that may lead to a change in the wafer lot sequence is the fact of having big transfer times, as it is shown in the following paragraphs. Let us consider the 6 × 5 problem instance, which is a small size example that was first solved under both scenarios using the robot movement data given in Tables 1 and 2. The solution structure, for both scenarios, corresponds to the 4–5–2–1–3 job sequence. The ERMN scenario rendered a makespan value of 72.29 time units (72 min, 17.4 s), and the ERMC one a performance measure of 73.57 (73 min, 34 s). This increase in the completion time is caused by the time the robot employs to make the empty trips.

**Table 6**
Enlarged loaded robot transfer times for the 6 × 5 example.

| *st*1 | *st*2 | *st*3 | *st*4 | *st*5 | *st*6 | *ob* |
|---|---|---|---|---|---|---|
| 1.2 | 0.6 | 0.8 | 1.0 | 0.4 | 0.6 | 1.0 |

To analyze the effect of an increase in the transfer times, the same 6 × 5 problem instance was solved with the loaded transfer time data presented in Table 6. Unloaded transfer times were calculated with the same rationale used before to build Table 1. This problem instance was again solved with both the ERMN and ERMC scenarios. The first scenario rendered a solution having a makespan value of 81.60 time units (81 min, 36 s). The wafer lot optimal sequence was 4–5–2–1–3 – the same obtained with lower values of the transfer times – and it was instantiated in less than 1 s. On the contrary, under the ERMC scenario, the 2–5–4–1–3 optimal sequence was obtained in 1866 s. Nevertheless, it is worth mentioning that despite the greater time needed to find the optimal solution, the first feasible schedule of the ERMC scenario, having a makespan of 128.70 time units (128 min, 42 s), was reached in only 4.8 s. This first solution is only 10.4% above the optimal one that has a makespan of 116.50 time units (116 min, 30 s).

The impact of the empty transfer times can be seen by comparing the Gantt charts shown in Fig. 8a and b, which correspond to the optimal solutions of the 6 × 5 example, with enlarged transfer times, under the ERMN and ERMC scenarios, respectively. For clarity purposes, in Fig. 8b, robot activities have been decoupled into loaded and unloaded transfer ones. As seen, empty movements are

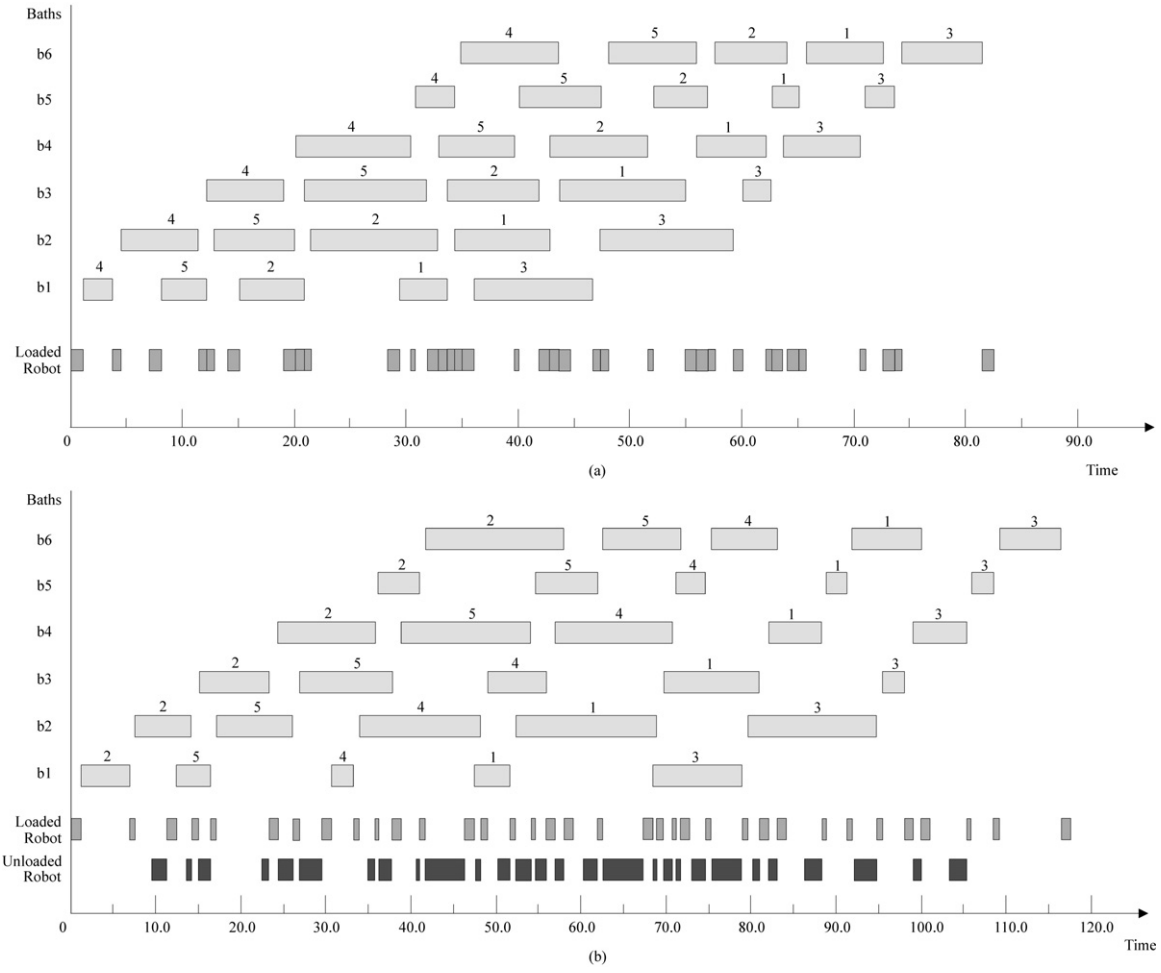


**Fig. 8.** Optimal schedules corresponding to the 6 × 5 case study with enlarged robot transfer times. (a) ERMN scenario. (b) ERMC scenario.

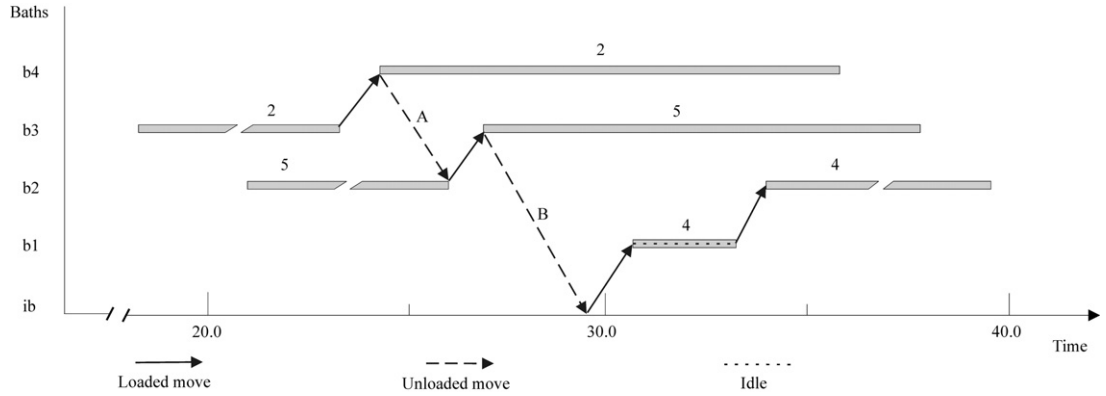


**Fig. 9.** Detailed view of the robot movements, both loaded and unloaded.

significant and they are responsible for introducing idle times on the processing stages. A more detailed view of these movements is presented in Fig. 9. This figure enlarges a small portion of the Gantt chart shown in Fig. 8b with the aim of illustrating the impact of the unloaded robot movements on the ERMC solution. Loaded robot movements are represented by solid arrows, while the unloaded ones correspond to dashed arrows. In addition, the time intervals the robot employs to make empty trips are identified with capital letters. The *A* interval represents the time required by the robot to go from bath *b*4, where it has already dropped wafer lot *j*2, to *b*2 in order to pick up wafer lot *j*5, which is then going to be transferred to bath *b*3. Similarly, interval *B* represents the time required by the robot to move without any load, from bath *b*3 to the *ib* input buffer to pick up wafer lot *j*4, which then needs to be transported to the first bath *b*1. After this, the device stays idle over this bath, waiting for the end of *j*4 processing. Then, it transports this wafer lot from *b*1 to *b*2. From this drawing and from Fig. 8b, it can be noticed that the times associated with the unloaded robot movements cannot be neglected if the chemical bath zero-wait policy is in fact to be enforced. In addition, it can be observed that these time intervals are of equal or bigger size than the ones corresponding to loaded trips.

### 4.2. Rolling horizon-based AWS operation. Insertion of new wafer lots

As it was described in Section 3.2, the AWS scheduling activity should not be addressed as an isolated problem. Automated wet-etch stations are part of the wafer fabrication stage, and are related with other preceding and succeeding manufacturing steps. Therefore, groups of new wafer lots continuously arrive at the AWS, turning its scheduling into an on-line activity, as it is shown in the example presented in the following paragraphs.

The AWS tackled in this case study consists of 6 stages with one bath per stage, an input and an output buffer, and one transfer device/robot. The example addresses a situation in which the AWS already has 5 scheduled jobs (*j*1–*j*5), and these wafer lots are currently in-progress in the unit. The makespan value of the on-going agenda is 73 min 41 s. Then, another set of 5 new jobs (*j*6–*j*10) is fed from photolithography and needs to be scheduled.

To illustrate the impact of a rolling horizon strategy, this case study has been solved under three different scenarios. (i) Scenario 1: The insertion time point is placed at the completion time of the last task belonging to the old agenda; thus, the new set of wafer lots is scheduled after all the AWS resources are available. This scenario is analyzed in order to show how the throughput of the station decreases if this policy were adopted. (ii) Scenario 2: The value of *it* is located as early as possible, and the in-progress and non-executed tasks belonging to the on-going schedule maintain their start times. (iii) Scenario 3: The value of *it* is located as early as possible, but some flexibility is associated with the in-progress and non-executed tasks. In fact, in-progress and non-executed rinsing tasks are allowed to increase their durations up to 20% ($\beta = 1.2$) and, consequently, non-executed etching tasks may modify their start and end times. Scenarios 2 and 3 illustrate the two alternative operating schemes that have been described in Section 3.2.

#### 4.2.1. Scenario 1

Fig. 10 depicts the Gantt chart corresponding to the optimal solution obtained for this case study, showing the insertion time

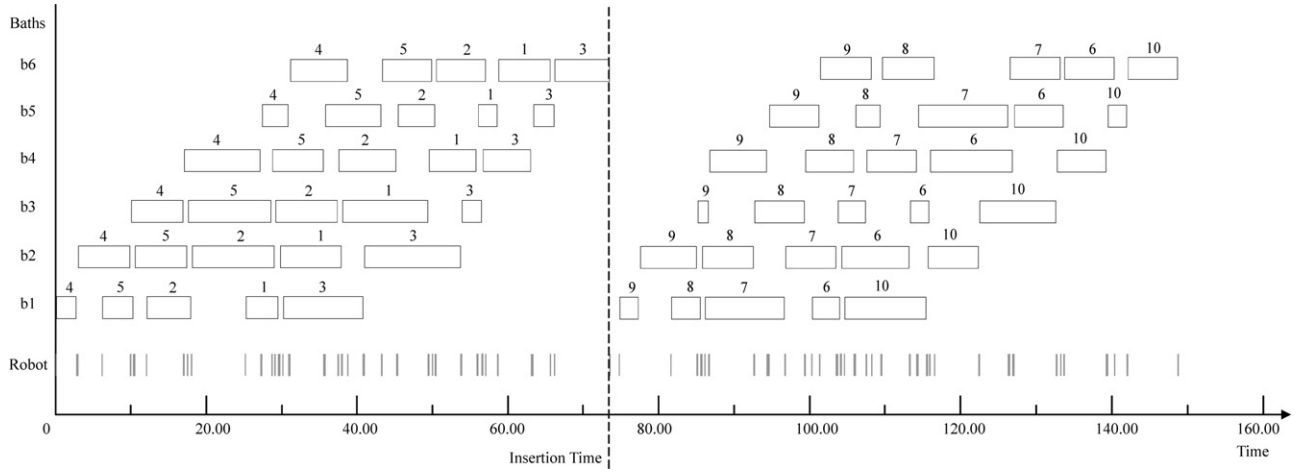


**Fig. 10.** Gantt diagram showing scenario 1 optimal solution.

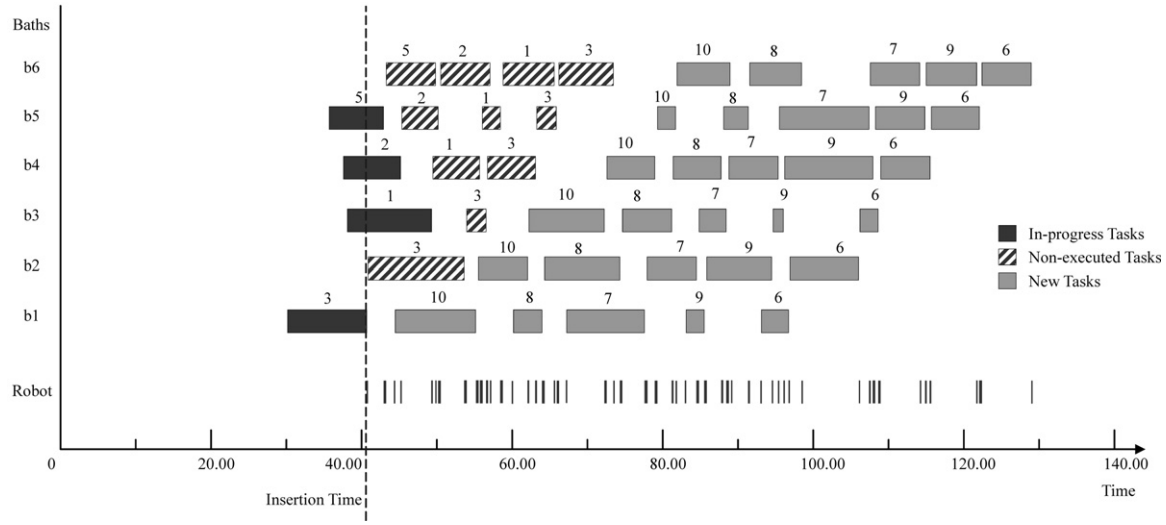


**Fig. 11.** Gantt diagram depicting scenario 2 optimal solution.

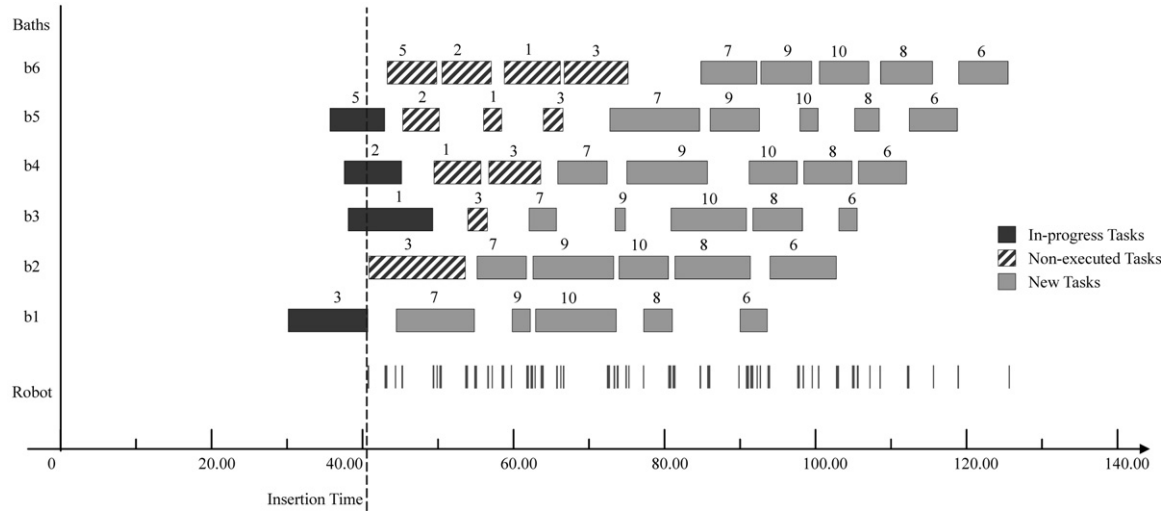


**Fig. 12.** Gantt diagram depicting scenario 3 optimal solution.

point associated with the end of the last processing task of the final wafer lot belonging to the first set to be scheduled. To solve this example the basic CP model was implemented and no limit on the solution time was imposed. The total makespan obtained after the insertion of wafer lots *j*6–*j10* is 148 min 52 s. The diagram illustrates the important non-working gap that appears between the two scheduling time horizons, i.e. the ones associated with the first and second sets of wafer lots. In this particular situation, bath idle times correspond to the void intervals that appear between the end of the tasks demanded by wafer lot *j*3 and the start of the activities required by *j*9. As seen, this type of operation would lead to a very low system throughput, since the average load of the various baths is very low.

Due to the situation described in previous paragraphs, this type of operation is by no means employed in practice. Nevertheless, it is the one that is implicitly assumed by other authors, which have always considered wet-etch scheduling in isolation and not as part of the wafer fabrication stage.

*4.2.2. Scenario 2*

Under this scenario the following values of the insertion time point and solution time window have been adopted: $it = 40$ min 49 s and $tw = 145$ s, respectively. The constraints that define the model of this problem instance are the ones of the basic CP model, plus constraints (19) and (24). Fig. 11 shows the schedule that was obtained within the solution time window *tw*. This solution was found in less than 1 s of CPU and proved to be optimal at 40 s. As prescribed, in-progress and non-executed tasks maintain their agenda. The total makespan corresponding to this solution is 129 min 4 s, which is almost 20 min lower than the corresponding one of scenario 1. The reasons for such large decrease are the big reduction on bath unproductive times and the good coupling between the old and the new agendas.

However, if the problem of scheduling wafer lots *j*6–*j*10 is analyzed in isolation, and the partial solutions of scenarios 1 and 2 are compared, it can be seen that whereas the first one employs 75 min 11 s to manufacture the new set of wafer lots, scenario 2 demands 88 min 15 s for the same duty. The 75 min 11 s partial makespan value is obtained by subtracting the makespan of the previous scheduling period to the total makespan. In turn, the 88 min 15 s partial makespan value corresponding to the second set of wafers in scenario 2 is attained by subtracting *it* to the new total makespan value. The reason for having this bigger value of the partial makespan is the fact that scenario 2 considers that the AWS

resources are not fully available when a new set of wafer lots needs to be scheduled. This analysis shows that seeking the best schedule for the new set of jobs without considering the current status of the AWS, as previous contributions in the field did, might be misleading. Nevertheless, it is important to notice that whereas the partial makespan of the second set of wafers in scenario 2 is worse than the corresponding one of scenario 1, its total makespan is much better, leading to a superior AWS performance in the long run.

#### 4.2.3. Scenario 3

The constraints that define the model of this problem instance are the ones of the basic CP model, plus constraints (20)–(24). Values of $\beta = 1.2$ have been adopted; thus, tasks associated with in-progress and non-executed rinsing operations are permitted to have durations of up to 20% longer.

The Gantt diagram depicted in Fig. 12 shows the optimal solution reached for this scenario, under the same *it* and *tw* (145 s) values adopted for scenario 2. This solution has a total makespan value of 125 min 44 s and it was found in 20 s. In order to assess its quality, the CPU time limit was disregarded. It was found that this is indeed the optimal solution; the optimality condition was proved at 210 s.

A comparison of Figs. 11 and 12 reveals that the total makespan is in this case around 4 min better than the one of the scenario 2 solution. This occurs because a better matching between the tasks that belong to the old and new agendas is now attained. The assessment of these solutions also discloses that the job sequences of the second set of wafers are different. The makespan reduction, as well as the change in the job order, takes place because of the additional flexibility given by means of the $\beta$ parameter (see expressions (20) and (22)). By allowing certain wafer lots to stay longer in their rising baths, the sequence of transfer operations can change to improve the station efficiency. Hence, scenario 3 shows the benefits of implementing a rolling horizon-based methodology that enables the AWS to operate in an almost continuous fashion. As seen, the new wafer lot set can be inserted in an efficient way that avoids having unproductive times between the two consecutive schedules and, thus, optimizes the total makespan as the global performance measure.

## 5. Conclusions

This contribution presents an innovative and general way of dealing with the automated wet-etch station scheduling problem, looking at some specific features that were not considered up to now. In this work the different types of robot movements have been studied in a comprehensive way, demonstrating the drawbacks that arise when unloaded robot trips are ignored and pointing out the limitations of previous approaches regarding this issue. It has been concluded that some of the optimal sequences reported by previous authors that ignored empty robot trips were unfeasible, leading in many cases to wafer damage. In addition, the operation of wet-etch stations within the context of semiconductor manufacturing process has been analyzed, and the importance of considering a rolling horizon-based operating policy has been highlighted.

Then, a CP formulation has been presented. This new model takes into account the loaded robot movements between consecutive baths, as well as the unloaded ones, by explicitly considering the time employed by the robot to travel empty from a bath, where it has already dropped a wafer lot, to another bath, where it needs to go in order to pick up a different lot. Afterward, the CP model has been generalized in order to implement a rolling horizon methodology that considers the current status of the AWS at the moment a new set of wafer lots is inserted in the on-going agenda.

Several case studies have been solved to illustrate the benefits of this proposal and to test it. Firstly, various examples of various dimensionalities have been solved under two circumstances: (i) without considering the empty movements of the robot, like previous contributions did, and (ii) taking them into account. The solutions found in the second situation, which obviously demanded more computational effort, have shown that the unloaded movements of the transfer device cannot be neglected; otherwise, the solutions that are obtained cannot be implemented in practice. Moreover, the wafer lot sequences that were reached when the unloaded robot trips were taken into account were, in many cases, different than the ones obtained when the empty robot movements were neglected. This occurs for big size problems (many wafer lots to be scheduled and/or a big number of baths), or when transfer times are important. The reported results have proved that the void robot trip simplifying assumption is incorrect. Thus, the problem cannot be tackled in a two-step solution approach in which the schedule sequence is sought first with a simplified model that ignores unloaded trips and, then, the AWS agenda is readjusted to include these trips.

Furthermore, the CP model extensions that implement the rolling horizon policy have been applied to another example. This case study demonstrates that AWS scheduling problems should not be addressed in isolation, as previous authors have done. On the contrary, they need to be tackled under a rolling horizon-based type of approach in order to obtain schedules that can be implemented in practice and that can improve the productivity of the manufacturing environment.

## Acknowledgements

The authors wish to acknowledge the financial support received from CONICET (PIP 2754), UNL (CAI+D 2009 R4 N12), and ANPCyT (PAE-PICT-2007-00051).

## References

Aguirre, A. M., & Méndez, C. A. (2010). A novel optimization method to automated wet-etch station scheduling in semiconductor manufacturing systems. In S. Pierucci, & G. Buzzi Ferraris (Eds.), *Computer-aided chemical engineering* (pp. 883–888). Elsevier Science.

Aguirre, A. M., Méndez, C. A., & Castro, P. M. (2011). A novel optimization method to automated wet-etch station scheduling in semiconductor manufacturing systems. *Computers & Chemical Engineering*, *35*, 2960–2972.

Bang, J.-Y., & Kim, Y.-D. (2011). Scheduling algorithms for a semiconductor probing facility. *Computers & Operational Research*, *38*, 666–673.

Bhushan, S., & Karimi, I. A. (2003). An MILP approach to automated wet-etch station scheduling. *Industrial and Engineering Chemistry Research*, *42*, 1391–1399.

Bhushan, S., & Karimi, I. A. (2004). Heuristic algorithms for scheduling an automated wet-etch station. *Computers & Chemical Engineering*, *28*, 363–379.

Che, A., Chabrol, M., Gourgand, M., & Wang, Y. (2012). Scheduling multiple robots in a no-wait re-entrant robotic flowshop. *International Journal of Production Economics*, *135*, 199–208.

Dabbas, R. M., Chen, H.-N., Fowler, J. W., & Shunk, D. (2001). A combined dispatching criteria approach to scheduling semiconductor manufacturing systems. *Computers & Industrial Engineering*, *39*, 307–324.

Dabbas, R. M., & Fowler, J. (1999). *A new scheduling approach using combined dispatching criteria in wafer fab operations*. Technical report IE-ORPS-99-001. Tempe, AZ: Department of Industrial Engineering, Arizona State University.

Geiger, C. D., Kempf, K. G., & Uzsoy, R. (1997). A tabu search approach to scheduling an automated wet etch station. *Journal of Manufacturing Systems*, *16*(2), 102–116.

Hung, Y.-F., & Wang, Q.-Z. (1997). A new formulation technique for alternative material planning – An approach for semiconductor bin allocation planning. *Computers & Industrial Engineering*, *32*(2), 281–297.

Johri, P. K. (1993). Practical issues in scheduling and dispatching in semiconductor wafer fabrication. *Journal of Manufacturing Systems*, *12*, 474–485.

Kallrath, J., & Maindl, T. I. (2006). *Real optimization with SAP APO* (pp. 105–118). Berlin: Springer-Verlag.

Karimi, I. A., Tan, Z. Y. T., & Bhushan, S. (2004). An improved formulation for scheduling an automated wet-etch station. *Computers & Chemical Engineering*, *29*, 217–224.

ILOG. (2002). *ILOG OPL Studio 3.7.1 user's manual*. France: ILOG.

ILOG. (2000a). *ILOG Scheduler 6.1 user's manual*. France: ILOG.

ILOG. (2000b). *ILOG Solver 6.1 user's manual*. France: ILOG.

Lee, Y., Kim, S., Yea, S., & Kim, B. (1997). Production planning in semiconductor wafer fab considering variable cycle times. *Computers & Industrial Engineering*, *33*(3–4), 713–716.

Manier, M.-A., & Bloch, A. (2003). A classification for hoist scheduling problems. *International Journal of Flexible Manufacturing Systems*, *15*(1), 37–55.

Nishi, T., Hiranaka, Y., & Grossmann, I. (2011). A bilevel decomposition algorithm for simultaneous production scheduling and conflict free routing for automated guided vehicles. *Computers & Operations Research*, *38*, 876–888.

Novas, J. M., & Henning, G. P. (2011). A novel CP approach for scheduling an automated wet-etch station. In C. Kiparissides, M. Georgiadis, & A. Kokossis (Eds.), *Computer-aided chemical engineering* (pp. 1085–1089). Elsevier Science.

Novas, J. M., & Henning, G. P. (2010). Reactive scheduling framework based on domain knowledge and constraint programming. *Computers & Chemical Engineering*, *34*, 2129–2148.

Pfund, M., Manson, S., & Fowler, J. W. (2006). Dispatching and scheduling in semiconductor manufacturing. In J. Herrmann (Ed.), *Handbook of production scheduling* (p. 213). Heilderberg: Springer.

Zeballos, L. J., Castro, P. M., & Méndez, C. A. (2011). Integrated constraint programming scheduling approach for automated wet-etch station in semiconductor manufacturing. *Industrial and Engineering Chemistry Research*, *50*, 1705–1715.