An optimization-based framework for the scheduling of Automated Manufacturing Systems

Adrián M. Aguirre*, Carlos A. Méndez*, Cesar De Prada**

* INTEC (UNL - CONICET), Güemes 3450, 3000 Santa Fe, Argentina, (Tel: +54-342-4559175; e-mail: cmendez@intec.unl.edu.ar).
**Universidad de Valladolid, c/ Real de Burgos s/n, 47011 Valladolid, Spain, (e-mail: prada@autom.uva.es).

Abstract: Automated Wet-Etch Station (AWS) is a complex flow shop operation process in Semiconductor Manufacturing Systems. In this station, automated material-handling robots are used to move wafer lots across a lineal configuration of chemical and water baths. In every bath, limited processing times and complex storage policies must be assured. In this work, an optimization-based framework is developed to improve the operations of AWS. To do this, a sequential procedure based on mixed-integer linear programming (MILP) formulations is proposed. The aim of this work is to provide a robust approach to generate near-optimal results to industrial AWS scheduling problems with modest CPU time.

1. INTRODUCTION

The scheduling of automated flow shop systems represents one of the most complex problems in literature nowadays. Many works have been developed in this area in the last 20 years in order to find reliable approaches for real-world large-scale scheduling problems. Despite the significant advances achieved, this challenging research area still remains without robust methodologies to obtain efficient solution to complex industrial problems with a reasonable computational performance (Ku and Karimi, 1990; Méndez et al., 2006).

Automated Wet-Etch Station (AWS) is a particular stage in the wafer fabrication process in semiconductor manufacturing systems. Processing operations in the AWS represent a particular flow shop production system in which several jobs have to be produced in serial single-unit stages in a sequential way (see Fig. 1). In this system, single arm robots perform the transportation activities of wafer's lots between consecutives stages (Uzsoy et al., 1992).

Fig. 1. Flow shop process operation scheme in AWS

The scheduling problem of processing and transportation activities taken into account in the AWS has been studied in detail in recent years. Different methodologies, such as exact mathematical formulations (Bhushan and Karimi, 2003; Aguirre et al., 2011a; Castro et al., 2012), constraint programming techniques (Zeballas et al., 2011), heuristics and meta-heuristic procedures (Geiger et al., 1997; Bhushan and Karimi, 2004), hybrid methods (Castro et al., 2011; Aguirre et al., 2012) and simulation tools (Aguirre et al., 2011b), have been developed to this problem. Many of these approaches are used to find optimal solutions to academic examples but others have been focused on generating effective results for industrial-sized cases with an acceptable computational cost.

In this work, a novel rigorous approach, by using a combination of MILP optimization models for the scheduling of multiple jobs in the AWS, is developed. The main contribution is to find an easy and a systematic methodology that allows generating, testing and also evaluating different solutions of industrial problems with short CPU time.

2. PROBLEM DEFINITION

Automated Wet-Etch Station (AWS) represents a complex multiproduct multistage batch manufacturing process. In this station, a set of different jobs or wafer lots \( i = 1, \ldots, N \) must be produced in several stages \( j = 1, \ldots, M \) of the process following the same manufacturing recipe. Every stage of this process is composed by a single unit, also called bath. Two different types of baths are common in this station: chemical baths \( j_{\text{ch}} = 1, 3, 5, \ldots, M-1 \) and water baths \( j_{\text{wa}} = 2, 4, 6, \ldots, M \). In chemical baths, wafer's lots are immersed in an abrasive liquid solution. These solutions allow removing the outer layer of the wafer, which is the principal aim of the process. After a wafer's lot leaves the chemical bath, traces of the substance remained in the wafer must be removed to avoid undesirable contamination between chemical baths. To do this, a water bath follows every chemical bath in order to clean up the pieces. Thus, water baths are disposed to rinse the wafer, leaving it ready for the next immersion process in the following stages.

In general, AWS presents a linear configuration of baths along the process. Thus, all jobs must be produced in the system by following this recipe \( j = 1, 2, 3, \ldots, M \). In addition, the structure of the process is composed by buffers at the beginning and at the end of the production line. Each buffer, Input Buffer \( j = 0 \) and Output Buffer \( j = M+1 \), keeps the...
wafer's lots before and after being processed. These are the only existing buffers in the entire process. Intermediate storage areas between consecutive baths are not considered.

Other important feature of this station is related to the transportation of wafer lots throughout the system. The transfer activities of jobs in the system are performed by automated material-handling devices called robots \((r=1,\ldots,M+1).\) These single-arm robots are in charge of all the wafer movements between consecutive stages, from bath \(j-1\) to bath \(j,\) and also from Input buffer \((j=0)\) to the first chemical baths \((j=1)\) and from the last baths \((j=M)\) to the Output buffer \((j=M+1)\) (see Fig. 2).

![Fig. 2. Lineal configuration of baths in AWS process structure](image)

A particular complexity of this system arises from the mixed intermediate storage policies \((SP)\) applied to every process stage. In this system, lots have to be processed without being discarded. One of the principal causes of discarding is associated with the overexposure time of wafer lots in the chemical reactive. To avoid this, every job must be immersed in chemical baths only during processing time. Extra residence time in chemical baths can damage the wafer's lot.

On the other hand, the water cannot damage the wafer's surface so jobs can be delayed in water baths without any restriction. According to this, Zero Wait storage policies \((ZW)\) must be strictly satisfied in chemical baths while Local Storage policies \((LS)\) should be allowed in water baths.

However, the most important limitation associated with this problem is the lack of intermediate storage between successive stages. This restriction must be fulfilled in the system by taking into account non-intermediate storage policies \((NIS)\) in each robot. Thus, robots cannot hold the wafer's lots more than the transferring time needed to move lots from one bath to another, in order to avoid damages in the final quality of wafers.

As a consequence, this problem represents a particular case of flow shop scheduling problems with mixed intermediate storage policies \((ZW/LS/NIS)\) and resource limitations. Also, a single robot has been considered in this problem to perform all transfer activities in the system.

In this work, an efficient solution strategy is proposed by using a suitable combination of existing optimization approaches. The main ideas of these approaches are explained in detail in the rest of the paper.

3. PRINCIPAL APPROACHES

3.1 The Slack time minimization Model

The Slack time minimization model is a modified version of a previous model proposed by Birewar and Grossmann (1989). The original model was developed for the resolution of flow shop scheduling problems. In that problem, several batches from different families of products have to be scheduled in a serial single-unit stages under strict zero-wait storage policies \((ZW),\) in order to determine the best sequence that minimize the total processing time \((MK)\) of all jobs in the system.

The MILP formulation introduced in that work used the ideas of slack time minimization in the objective function of the model. The slack time among these batches is strongly related to the order in which these ones are sequenced. Thus, two batches should be sequenced one after the other only if the slack time between them is minimum.

The problem addressed in that work is quite similar to the one presented here. In AWS scheduling problem several jobs of different families have to be sequenced in a lineal configuration of single-unit stages under strict intermediate storage policies between consecutives units. Also, transfer activities must be performed by a single-arm robot, as an automated transportation device, which is in charge to move these jobs in the entire process.

The principal difference between these two problems relies in the mixed storage policies and also in the transfer activities considered. Despite of this, the original MILP approach of Birewar and Grossmann (1989) can be easily adapted to comprise some of the principal features of AWS scheduling problem with the existing formulation.

In this work, a rigorous approach for the AWS scheduling is presented. The proposed MILP formulation tries to minimize the makespan criterion considering the slack time minimization and the transferring tasks between consecutive units.

The main limitation of this new formulation is the use of additional binary variables (immediate precedence variables) to achieve good job sequencing decisions of the problem. However, this limitation is mitigated by incorporating the slack calculation derived from Birewar and Grossmann (1989) into the objective function of the problem.

Transfer times will be considered into the slack parameter, supposing that unlimited robots are available to perform all the transfers in the system. The calculation of this parameter requires that zero-wait storage policies will be ensured in the entire process. Thus, the slack time \((SL_{i,j})\) will be estimated as the idle time of two consecutives jobs \(i\) and \(i'\) in an specific unit \(j,\) as it can be seen in Fig. 3.

![Fig. 3. Flow shop scheduling problem for \(N\) jobs in \(M\) units with unlimited resources for transfer operations](image)
According to this, the new formulation provides a constrained representation of the unlimited robot model (URM) developed by Bhushan and Karimi (2003), for the problem presented in this work. The main advantage of the Slack time model is that it allows solving this kind of NP-hard permutation problem in short computational time (Ku and Karimi, 1990), giving alternative job’s sequences with minimum makespan (MK).

3.1.1 Slack time MILP-based formulation

The exact mathematical formulation presented below is based on the main ideas of immediate precedence concepts (see the review of Méndez et al., 2006). Some important decision variables are defined as follows: \( Z_{ij} \) denotes that job \( i \) is the first job in the processing sequence (\( Z_{ij}=1 \)) while \( F_{ij} \) means that job \( i \) is the last job produced (\( F_{ij}=1 \)). Variable \( X_{ij} \) represents that job \( i \) is visited right before job \( i' \) in the processing sequence by adopting value 1. In other cases, all these variables will adopt value 0. Also, continuous variables Position\((i)\) and MK are proposed to represent the allocation of each job \( i \) in the processing sequence and the completion time of all the jobs in the system respectively.

The model proposed comprises the following equations (1)-(10). Equations (1)-(6) are defined under immediate precedence concepts to provide feasible job's sequences without generating sub-tours. Thus, (6) is applied to determine the exact allocation of a particular job \( i \) in the processing sequence. And (7) is established, for the makespan calculation (MK) by using all the information obtained from the sequencing variables and also from the slack parameter. In addition, constraint (19) is proposed to provide alternative job sequences over time.

Parameters, \( Dif, Violation \) and \( SL \) are defined beforehand in (8)-(10) by the information of the production time \( t_{ij} \) and transferring time \( \pi_i \) of every job \( i \) in every unit \( j \). Parameter \( Dif_{ij} \) represents the differences between the initial and the final time of dissimilar jobs \( i \) and \( i' \) in the stage of unit \( j \). \( Violation_{ij} \) parameter determines the time violation of these two jobs \( i \) and \( i' \), as the minimal value of \( Dif_{ij} \) parameter for any unit \( j \). Finally, the slack time \( SL_{ij} \) of jobs \( i, i' \) in unit \( j \) is calculated by the difference between \( Dif_{ij} \) and \( Violation_{ij} \).

### Determining the antecessor of job \( i \)

\[
\sum_{j \in I} X_{ij} + Z_{ij} = 1 \quad \forall i \in I \tag{1}
\]

### Determining the successor of job \( i \)

\[
\sum_{j \in I} X_{ij} + F_{ij} = 1 \quad \forall i \in I \tag{2}
\]

### Avoiding sub-cycle constraints.

\[
Z_{ij} + F_{ij} \leq 1 \quad \forall i \in I \quad \sum_{i} Z_{ij} = 1 \quad \sum_{i} F_{ij} = 1 \tag{3-5}
\]

### Position\((i') \geq Position\((i) +1 - N^\pi (1 - X_{ij})\)\)

\[
Position(i') \geq Position(i) + 1 - N^\pi (1 - X_{ij}) \tag{6}
\]

### Slack Calculation.

\[
Dif(i,i',j) = \sum_{j \in I} X_{ij} (F_{ij} + \pi_i) - \sum_{j \in I} X_{ij} (F_{ij} + \pi_{i'}) \tag{7}
\]

\[
Violation(i,i') = \min_{j} (Dif(i,i',j) - \pi_{j + 1}) \quad \forall i, i' \in I : (i \neq i') \tag{8}
\]

### SL Calculation.

\[
SL(i,i',j) = |Dif(i,i',j) - Violation(i,i')| \quad \forall i, i' \in I : (i \neq i') \tag{9}
\]

3.2 The Unlimited Robot Model (URM)

The Unlimited Robot Model (URM) was initially proposed by Bhushan and Karimi (2003) for the MK minimization under a slot-based formulation and later it was reformulated by Aguirre et al. (2011) by using the features of general precedence concepts (Méndez et al. 2006). The last formulation provides a reduced model representation in terms of binary variables for job sequencing decisions. Here, the exact mathematical model of Aguirre et al. (2011) under general precedence ideas is used.

The principal advantages of this robust MILP model is that it provides near-optimal solutions to AWS scheduling problems in a reasonable computational time, without considering resource limitations. Also, this model can be used as a LP formulation by fixing the \( X_{ij} \) binary variables when we have prior information of the system. The values of these variables can be defined in advance by solving first the slack model presented above. The suitable combination of these methods allows generating near-optimal job sequences for industrial-sized AWS problems in short CPU time, by considering unlimited number of robots available.

3.2.1 Relaxed MILP-based formulation

The relaxed model presented in this work provides a more realistic representation of the real-life AWS scheduling problem, than the Slack time model, by considering specific intermediate storage policies (SP) in every unit \( j \). Thus, zero-wait (ZW) storage policies must be applied only in odd units \( j_{odd}=1,3,5,...,M+1 \) while local storage policies (LS) are allowed in even units \( j_{even}=2,3,6,...,M \), as it shows in (11)-(12). In the same way, due to the non-existence of buffers between consecutive units, non-intermediate storage policies (NIS) are imposed in every unit by (13)-(14).

Despite of this, the proposed formulation considers that unlimited robots are available at any time to perform transfer activities. Thus, equations (15)-(16) are used to determine the correct synchronization of different jobs in the system. For this, continuous variables \( T_{x_{ij}} \) and \( T_{f_{ij}} \) are defined as the start time and the completion time of every job \( i \) in every unit \( j \). Also, \( MF \) parameter is proposed as a large number in order to ensure the conditions expressed in equations (15) and (16).
Job sequencing variables $X$ are calculated in (17) by the Position parameter previously defined. Finally, the calculation of the makespan is reported in (18) while an additional constraint presented in (19) is used to generate alternative and always feasible job sequences of the system. Here is worth to remark that equation (19) is defined as an integer cut, providing alternative sequences of the system without generating unfeasible results.

Timing constraints.

$$T_{f(i,j)} = T_{s(i,j)} + t_{i,j} \quad \forall i \in I, j \in J_{odd}, SP_j = ZW$$  \hspace{1cm} (11)

$$T_{f(i,j)} \geq T_{s(i,j)} + t_{i,j} \quad \forall i \in I, j \in J_{even}, SP_j = LS$$  \hspace{1cm} (12)

Transfers constraints.

$$T_{s(i,j)} = T_{f(i,j-1)} + \pi_j \quad \forall i \in I, j \in J : j > 1, SP_j = NIS$$  \hspace{1cm} (13)

$$T_{s(i,j)} \geq \pi_j \quad \forall i \in I, j = 1$$  \hspace{1cm} (14)

Job sequencing constraints.

$$T_{s(i,j)} \geq T_{f(i,j')} + \pi_j + \pi_{j-1} - M \tau(1 - X_{(i,j')})$$  \hspace{1cm} (15)

$$\forall i, i' \in I : (i > i'), j \in J$$

$$T_{s(i,j)} \geq T_{f(i,j')} + \pi_j + \pi_{j-1} - M \tau(X_{(i,j')})$$  \hspace{1cm} (16)

$$\forall i, i' \in I : (i > i'), j \in J$$

Fixing job’s sequencing decisions.

$$X_{(i,j')} = \begin{cases} 1 & \text{if Position (i) > Position (i')} \\ 0 & \text{if Position (i) < Position (i')} \end{cases}$$  \hspace{1cm} (17)

Makespan Calculation.

$$MK \geq T_{s(i,j)} \quad \forall i \in I, j = M + 1$$  \hspace{1cm} (18)

Additional cuts for alternative job’s sequences.

$$\left( \sum_{i} \sum_{j} X_{(i,j')} \right) \leq \left( \sum_{i} \sum_{j} 1 \right) + \left( \sum_{i} \sum_{j} X_{(i,j')} \right) - 1$$  \hspace{1cm} (19)

3.3 The One Robot Model (ORM)

The ORM model presented here is a simplified version of the one developed by Aguirre et al. (2011a). In this work only one robot are considered for the wafer movements. The correct synchronization of transport activities can generate high computational costs. Due to this, great efforts have been done in the way to reduce the level of decisions involved in the system and also the CPU time required to solve the whole problem in a simple manner.

3.3.1 Reduced MILP-based formulation

The reduced MILP-based model is proposed to solve the entire problem considering job's sequencing decisions $X$ in (15)-(16) and transfer’s sequencing decisions $Y$ as it is expressed in (20)-(21). On the other hand, (22) is presented to reduce the number of transfer’s sequencing decisions to a manageable level.

Transfer’s sequencing constraints.

$$T_{s(i,j)} \geq T_{s(i,j')} + \pi_j - M \tau(1 - Y_{(i,j,j')})$$  \hspace{1cm} (20)

$$\forall i, i' \in I : (i > i'), j, j' \in J$$

$$T_{s(i,j')} \geq T_{s(i,j)} + \pi_j - M \tau(Y_{(i,j,j')})$$  \hspace{1cm} (21)

Fixing transfer’s sequencing decisions.

$$Y_{(i,j,j')} = \begin{cases} 1 & \text{if Position (i) > Position (i') \land j > j'} \\ 0 & \text{if Position (i) < Position (i') \land j < j'} \end{cases}$$  \hspace{1cm} (22)

4. MAIN CONTRIBUTIONS

4.1 Sequential solution strategies

In this section, two novel strategies developed for the AWS scheduling problem are exposed. The whole problem is decomposed in different sub-problems that must be solved sequentially. The sequential procedure utilized in this work is based on the ideas of Bhushan and Karimi (2003). In that work, they propose a two-stage algorithm for the AWS scheduling problem. In the first stage of this algorithm, a simplified problem is solved without considering robot limitations. After that, a feasible sequence of jobs is generated. In the second stage, the restrictions of a single robot are taking into account. Finally, a detailed processing and transfer schedule of the problem is determined by using the fixed sequence provided in the first stage.

For each of these sub-problems, a particular model is used. In there, an Unlimited Robot Model (URM) is proposed to handle job sequenced decisions without considering robot limitations while the One Robot Model (ORM) is considered for the resolution of the whole problem taking into account single-robot constraints. This bi-level strategy, called Robot-Constrained Unlimited Robot Model (RCURM), was proposed for solving the entire problem in a sequential way.

In our sequential strategy (URM-ORM), the solution of the first sub-problem is obtained by solving the URM model in (11)-(16) and (18)-(19), in order to find a feasible job sequence in each iteration. Then, in the second stage algorithm, a feasible schedule of the whole system is provided by ORM model in (11)-(18) and (20)-(22) (Fig. 4).
In the second procedure, the best feasible job sequences of the AWS scheduling problem, is obtained by using a combination of optimization methods. For this, Slack time and URM models are solved sequentially in order to find alternative sequences of the relaxed problem. Here, the Slack time MILP-based model, in (1)-(11) and (19), is solved at first to find the permutation sequence with the minimal MK$_{SLACK}$, by considering ZW policies between every processing unit. The solution reported by this model provides an upper bound for the resolution of the subsequent URM model. After that, the URM model is solved, as a LP model in (11)-(18), by fixing the job’s sequencing decisions, in order to find a detailed schedule and a real MK$_{URM}$ of the relaxed problem. In the second step, the ORM model is solved to find a feasible robot’s schedule of the entire system that minimize the MK$_{ORM}$, by using (11)-(18) and (20)-(22) (Fig. 5).

Fig. 5. Detailed optimization algorithm (Slack-URM-ORM)

Once the solution of the first stage is obtained, additional cuts are applied in successive iterations, by adding (19), in order to find alternative job’s sequences with the best relaxed MK.

The feasibility of the solution found is referred to the unconstrained case provided by URM or Slack model. In here, robots restrictions are not taken into account and continuous variables $T_{S(i,j)}$ and $T_{F(i,j)}$ are set free in each iteration, allowing the easy implementation of mixed storage policies of the system by the MILP model.

5. RESULTS AND COMPARISONS

5.1 Examples

The following problems (P1-P10) to be tested were obtained from the information of the first $M$ baths and $N$ jobs of the example provided by Bhushan and Karimi (2004). Many moderate size and industrial size cases are analyzed by using 8 and 12 baths and several jobs. The largest industrial problem (P10) is considered including 12 baths (processing units) and 20 different jobs.

5.2 Results

The solution strategy of this work aims towards enhancing the results provided by the full-space model and the sequential procedures RCURM of Aguirre et al. (2011a) and also reducing the computational time consumed by hybrid methods presented by Castro et al. (2011) and Aguirre et al. (2012). The statistics of the problems analyzed are summarized in Table 1.

Table 1 demonstrates that the sequential use of URM-ORM models provides better results than the Slack-URM-ORM approach for the first six moderate-sized cases analyzed. This happens because the exact URM model is able to find better sequences than Slack-URM models in a low number of iterations. As an example, Fig. 6 shows the behavior of the URM-ORM sequential approach for $MN=12x8$ problem instance analyzed (P3). In this instance, the solution of the URM model and ORM converge to the best result (MK$_{ORM}=$170.6 units) after ten iterations. Here, we can easily observe that the best URM sequence (MK$_{URM}=$170.3 units) not always generate better ORM results. Then, by exploring alternative sequences, we can find better solutions with reasonable computational effort. Figure 6 also shows the CPU time of each model per iteration and the total accumulative CPU time in seconds.

Fig. 6. Behavior of the URM-ORM solution approach for P3.

However, when the problem size increases, the Slack-URM-ORM performs better, providing good-quality solutions in short CPU time. Although the computation time is reduced, the Slack-URM procedure needs much more iterations to reach better initial sequences than the exact URM model.

Despite of the good performance, only the Slack-URM-ORM procedure can provide a feasible result (MK$_{ORM}=$340.8 units) for the largest problem analyzed (P10), within a time limit imposed of 3600 seconds. In order to overcome this computational limitation, the ORM model may be changed by a decomposition method or by a heuristic-based algorithm, which allows reporting feasible initial results with a reasonable CPU burden.

5.3 Comparisons

A comprehensive comparison with other existing techniques applied to this complex optimization problem, is reported in Table 2. This table shows de efficiency of the solution approach developed in this work. The proposed strategies report better results in some medium-sized cases (P1-P4) than the original RCURM and the full-space ORM model also with less CPU time. For many industrial-sized problems (P5-P10) the solution of our methods are comparable with the best results provided by the CP approach of Zeballos et al. (2011), the ORM-Hybrid Sim./Opt. approach of Castro et al. (2011b), and the ORM-Hybrid MILP-based decomposition approach of Aguirre et al. (2012).

Table 2. This table shows de efficiency of the solution approach developed in this work. The proposed strategies report better results in some medium-sized cases (P1-P4) than the original RCURM and the full-space ORM model also with less CPU time. For many industrial-sized problems (P5-P10) the solution of our methods are comparable with the best results provided by the CP approach of Zeballos et al. (2011), the ORM-Hybrid Sim./Opt. approach of Castro et al. (2011b), and the ORM-Hybrid MILP-based decomposition approach of Aguirre et al. (2012).
As an example, for instance P9, our URM-ORM procedure provides a solution of MK\_ORM = 256.5 units in 601 seconds when the best result reached by the hybrid technique of Castro et al. (2011b) was MK\_ORM = 260.2 units after 2275 seconds and the result obtained by the solution approach of Aguirre et al. (2012) was MK\_ORM = 268.2 units in 867 seconds.

6. CONCLUSIONS

Two novel optimization-based methods have been developed to simultaneously address the scheduling problem of manufacturing and material-handling operations in semiconductor industry. The proposed approaches can be easily used to test and improve different schedules. We illustrate that the URM-ORM solution approach is able to generate very effective results to moderate-sized problems with modest computational effort in comparison with other existing techniques. Also, the Slack-URM-ORM procedure has been very useful to find alternative feasible solutions to the whole problem, in some industrial-sized cases, with a reasonable computational effort (< 3600 sec.).

ACKNOWLEDGMENTS

Financial support received from AECID under Grant PCI-A1/044876/11 and from UNL under Grant PI-66-337 is fully appreciated.

REFERENCES


Table 1. Statistics and best results of the proposed solution methods applied to AWS scheduling problems

<table>
<thead>
<tr>
<th>P</th>
<th>MsN</th>
<th>MK_ORM</th>
<th>MK_ORM</th>
<th>CPU (sec.)</th>
<th>TCPU (sec.)</th>
<th>Nº iter.</th>
<th>Max. iter.</th>
<th>Max. time /model</th>
<th>Total time /iter</th>
<th>MK_ORM</th>
<th>MK_ORM</th>
<th>CPU (sec.)</th>
<th>TCPU (sec.)</th>
<th>Nº iter.</th>
<th>Max. iter.</th>
<th>Max. time /model</th>
<th>Total time /iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>8x10</td>
<td>149.5</td>
<td>152.9</td>
<td>19.8</td>
<td>135</td>
<td>2</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>151.5</td>
<td>153.5</td>
<td>4.1</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>P2</td>
<td>12x8</td>
<td>170.6</td>
<td>170.6</td>
<td>15.4</td>
<td>20</td>
<td>8</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>170.6</td>
<td>174.2</td>
<td>1.9</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>P3</td>
<td>12x10</td>
<td>192.4</td>
<td>196.2</td>
<td>88.2</td>
<td>324</td>
<td>2</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>194.2</td>
<td>196.5</td>
<td>43.9</td>
<td>44</td>
<td>10</td>
<td>10</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>P4</td>
<td>12x12</td>
<td>210.7</td>
<td>215.8</td>
<td>180.8</td>
<td>404</td>
<td>4</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>215.8</td>
<td>219.5</td>
<td>155.0</td>
<td>155</td>
<td>10</td>
<td>10</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>P5</td>
<td>8x18</td>
<td>230.4</td>
<td>240.6</td>
<td>157.2</td>
<td>346</td>
<td>3</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>232.6</td>
<td>241.2</td>
<td>72.46</td>
<td>73</td>
<td>10</td>
<td>10</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>P6</td>
<td>8x20</td>
<td>250.8</td>
<td>261.6</td>
<td>122.7</td>
<td>342</td>
<td>3</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>251.6</td>
<td>265.7</td>
<td>145.7</td>
<td>159</td>
<td>6</td>
<td>10</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>P7</td>
<td>8x22</td>
<td>277.1</td>
<td>289.8</td>
<td>354.1</td>
<td>478</td>
<td>8</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>277.8</td>
<td>288.2</td>
<td>126.4</td>
<td>140</td>
<td>7</td>
<td>10</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>P8</td>
<td>8x24</td>
<td>324.7</td>
<td>354.6</td>
<td>427</td>
<td>8</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>318.3</td>
<td>328.1</td>
<td>37.2</td>
<td>51</td>
<td>6</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>30</td>
</tr>
<tr>
<td>P9</td>
<td>12x15</td>
<td>244.9</td>
<td>256.5</td>
<td>421.2</td>
<td>601</td>
<td>7</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>246.1</td>
<td>260.0</td>
<td>181.2</td>
<td>603</td>
<td>3</td>
<td>10</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>P10</td>
<td>12x20</td>
<td>- NFS</td>
<td>- 3600</td>
<td>- 10</td>
<td>30</td>
<td>60</td>
<td>297.1</td>
<td>340.8</td>
<td>1200</td>
<td>3600</td>
<td>2</td>
<td>10</td>
<td>30</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 2. Results of different solution strategies summarized for many AWS scheduling problems instances analyzed

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>152.1</td>
<td>2113</td>
<td>- -</td>
<td>156.7</td>
<td>18.3</td>
<td>- -</td>
</tr>
<tr>
<td>P2</td>
<td>170.6</td>
<td>180</td>
<td>- -</td>
<td>171.3</td>
<td>2.53</td>
<td>- -</td>
</tr>
<tr>
<td>P3</td>
<td>202.2</td>
<td>3600</td>
<td>199</td>
<td>3600</td>
<td>197.2</td>
<td>42.8</td>
</tr>
<tr>
<td>P4</td>
<td>222.0</td>
<td>3600</td>
<td>- -</td>
<td>215.8</td>
<td>3857</td>
<td>- -</td>
</tr>
<tr>
<td>P5</td>
<td>260.1</td>
<td>3600</td>
<td>246.9</td>
<td>3600</td>
<td>240.4</td>
<td>3645</td>
</tr>
<tr>
<td>P6</td>
<td>279.5</td>
<td>3600</td>
<td>270.5</td>
<td>3600</td>
<td>262.1</td>
<td>3623</td>
</tr>
<tr>
<td>P7</td>
<td>307.1</td>
<td>3600</td>
<td>303.6</td>
<td>3600</td>
<td>289.9</td>
<td>3693</td>
</tr>
<tr>
<td>P8</td>
<td>348.7</td>
<td>3600</td>
<td>347.7</td>
<td>3600</td>
<td>331.5</td>
<td>3692</td>
</tr>
<tr>
<td>P9</td>
<td>NFS3</td>
<td>3600</td>
<td>273.2</td>
<td>3600</td>
<td>NFS3</td>
<td>7200</td>
</tr>
<tr>
<td>P10</td>
<td>NFS3</td>
<td>3600</td>
<td>353.2</td>
<td>3600</td>
<td>NFS3</td>
<td>1400</td>
</tr>
</tbody>
</table>

MK = Makespan, NFS = No feasible solution found. CPUs = Computational time to find the best result in seconds. TCPU = Total computational time of the algorithm in seconds. Nºiter = Number of iterations needed to find de best result. Max. iter. = maximum number of iterations. Max. time/model = maximum computational time for each model. Total time/iter. = Total computational time per iteration. Results obtained by using: (a) GAMS with Gurobi 6.0 in an Intel PC Core 2 Quad parallel processing in 4 threads (b) Termination criterion = 3600sec.

785