Solving an Avionics Real-Time Scheduling Problem by Advanced IP-Methods

Friedrich Eisenbrand[1], Karthikeyan Kesavan[2], Raju S. Mattikalli[2], Martin Niemeier[1], Arnold W. Nordsieck[2], Martin Skutella[3], José Verschae[3], and Andreas Wiese[3]

[1] EPFL, Lausanne, Switzerland
[2] Boeing, USA
[3] TU Berlin, Germany

Abstract We report on the solution of a real-time scheduling problem that arises in the design of software-based operation control of aircraft. A set of tasks has to be distributed on a minimum number of machines and offsets of the tasks have to be computed. The tasks emit jobs periodically starting at their offset and then need to be executed on the machines without any delay. Also, further constraints in terms of memory usage and redundancy requirements have to be met. Approaches based on standard integer programming formulations fail to solve our real-world instances. By exploiting structural insights of the problem we obtain an IP-formulation and primal heuristics that together solve the real-world instances to optimality and outperform text-book approaches by several orders of magnitude. Our methods lead, for the first time, to an industry strength tool to optimally schedule aircraft sized problems.

1 Introduction

Modern aircraft computing systems are employing new architectures that provide significant weight and cost advantages over previous architectures. They include computing, network, and I/O modules that are highly configurable. However these architectures present integration complexities which require automated methods to achieve configuration design centering. One such complexity is the allocation and scheduling of applications on processors. Due to the hard real-time nature of the airplane systems, a static cyclic execution schedule is required. For allocation purposes, applications are characterized by performance, memory, I/O, operational availability, functional separation, and functional grouping requirements. The processors are characterized by performance as implemented in a schedule, limited memory capabilities, and limited I/O capabilities via the network. A significant challenge associated with solving this problem in the context of a commercial aircraft is that of scale – it requires careful consideration of problem formulation and solution efficiency.

We report on integer programming approaches for this problem. It turns out that textbook formulations, like the time-indexed formulation, are not well suited to tackle the problems of complexity arising in real-world applications. By exploiting structural insights of the problem we provide an integer programming model and primal heuristics that outperform the textbook approach significantly. By restricting to a relevant subclass of instances and exploiting a *bin-tree* structure [EHN$^+$10], we obtain a model that is tailored to this subclass and outperforms the other formulations drastically. The latter

formulation is able to handle industrial size instances of the scheduling problem. Even for real-world instances not belonging to the subclass, one can still use the formulation as a heuristic using a rounding technique. For our real-world instance, this heuristic finally provides optimal solutions.

1.1 Problem Definition

Our problem is a variant of the *periodic maintenance problem* (PMP) [WL83]. First, we describe a simplified version called the *basic PMP*. It is the core of the real problem that Boeing is challenged with, the *extended PMP*, which we describe afterwards.

In the basic PMP we are given a set of tasks $\mathscr{T} = \{\tau_1, \ldots, \tau_n\}$ where each task $\tau_i = (c_i, p_i)$ is characterized by its *execution time* $c_i \in \mathbb{N}$ and *period* $p_i \in \mathbb{N}$. The goal is to assign the tasks to identical machines and to compute offsets $a_i \in \mathbb{N}_0$. A task τ_i generates one *job* with execution time c_i at every time unit $a_i + p_i \cdot k$ for all $k \in \mathbb{N}_0$. Each job needs to be processed immediately and non-preemptively after its generation on the task's machine. A *collision* occurs if two jobs are simultaneously active on the same machine. A schedule is feasible if no collision occurs. In the sequel, we denote by $Q = \{q_1, ..., q_k\}$ the set of all period lengths arising in the respective instance. We assume that $q_1 < \ldots < q_k$. An important special case, in particular in real-world instances, is the case of *harmonic periods*. In this case for each pair of tasks $\tau_i, \tau_{i'}$ we have that either $p_i | p_{i'}$ or $p_{i'} | p_i$.

In the extended PMP, machines have additional resource limitations in terms of memory of different types (RAM, ROM, etc.) and communication links that need to be considered. Each task has a given requirement for each type of memory and needs certain communication links to be open on its machine. Each machine can handle only a limited number of links and a limited bandwidth used by them. Like in the basic PMP, all machines are identical.

Moreover, due to system stability requirements certain tasks need to be assigned to different machines. Also, the machines have to be partitioned into two cabinets (left and right). To this end, we are given sets of tasks which have to be distributed evenly among the cabinets in order to design a fail-safe architecture.

1.2 Related Work

There is excessive literature on real-time scheduling; see, e.g., [BHR93,But04,Leu04] for surveys. The periodic maintenance problem was introduced by Wei and Liu [WL83] in the context of single machine and unit execution times. Baruah et al. [BRTV90] and independently Korst et al. [KALW91,KAL96] show that the periodic maintenance problem is NP-hard. Moreover, minimizing the number of machines is hard to approximate within a factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$ unless $P = NP$ [EHN$^+$10,Bha98].

The (basic) PMP generalizes BIN-PACKING. For Bin-Packing, First-Fit with decreasing item sizes is a 1.5-approximation algorithm [SL94] which is best possible unless $P = NP$. For the harmonic case of the PMP a First-Fit heuristic achieves an approximation factor of 2 and it is NP-hard to approximate it any better [EHN$^+$10]. The general case cannot be approximated non-trivially if $P \neq NP$ [EHN$^+$10].

From a computational point of view, there exists extensive literature on heuristics as well as branch-and-bound and column generations methods for Bin-Packing. In [MT90] polynomial time approximation algorithms as well as lower bounds, and exact algorithms are studied. Lower bounds to the optimal solution (which can be computed fast) are presented in [CPPT07]. Several approaches have been proposed for solving the Bin-Packing problem with branch-and-price techniques; see, e.g., [Van99,VBJN94,VdC99]. An algorithm called BISON is proposed in [SKJ97], where branch-and-bound techniques and tabu search are combined to design an exact hybrid algorithm. There is also a lot of literature on heuristics for the Bin-Packing problem. For example, Gupta and Ho propose a heuristic that greedily minimizes the slack of the machines. Fleszar and Hindi [FH02] modify these ideas and combine them with variable neighborhood search to design an hybrid algorithm. Moreover, Loh et al. [LGW08] propose a simple local heuristic based in the concept of *weighted annealing*.

2 Structural Insights

We now review some properties of the (basic) PMP which we will exploit later in our IP-formulations. First, we state a lemma which formulates an algebraic condition for the collision of two tasks, shown by Korst et al.

Lemma 1 ([KALW91]). *Let τ_i and $\tau_{i'}$ be two tasks which are scheduled on the same machine with offsets $a_i \in \mathbb{N}_0$ and $a_{i'} \in \mathbb{N}_0$, respectively. They do not collide if and only if*

$$c_{i'} \leq (a_i - a_{i'}) \bmod \gcd(p_i, p_{i'}) \leq \gcd(p_i, p_{i'}) - c_i.$$

We now restrict to the case of harmonic periods and describe some structural properties of this case. First, we sketch the concept of bin-trees which was first introduced in [EHN$^+$10].

Assume we have a feasible schedule for an harmonic instance of tasks $\tau_i = (c_i, p_i)$, $i = 1, \ldots, n$ on one machine, given by an offset a_i for each task. Due to a shifting argument we can show that there exists a feasible schedule in which a task τ_i with $p_i = q_1$ has offset $a_i = 0$. This task divides the *hyperperiod* $[0, q_k)$ (after which the schedule repeats itself) into *bins* $B_\ell = [\ell \cdot q_1, (\ell+1) \cdot q_1)$ with $\ell \in \{0, \ldots, q_k/q_1 - 1\}$. Using an exchange argument we can also show that w.l.o.g. inside each bin the jobs are ordered by the period length of the tasks which created them. Moreover, the jobs are executed consecutively and all idle time is accumulated at the end of the bin. See [EHN$^+$10] for formal proofs of the assumptions made. Figure 1 shows an example schedule with the described adjustments.

An important observation is the following: Consider two bins $B_\ell = [\ell \cdot q_1, (\ell+1) \cdot q_1)$ and $B_{\ell'} = [\ell' \cdot q_1, (\ell'+1) \cdot q_1)$ such that $\ell \equiv \ell' \bmod q_r/q_1$. As far as tasks with period length up to q_r are concerned, these bins look the same. Hence, the whole structure can be represented as a tree (see Figure 2). Each node in level ℓ encodes the tasks of period length up to q_ℓ that are scheduled in all its child nodes.

From any feasible schedule with the structure described above, we can obtain an assignment of tasks to the bins of a machine. We show in the following lemma that the

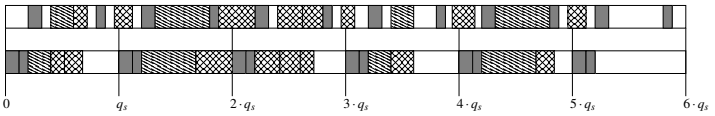

Figure 1. A schedule for a single machine and a schedule for the same tasks which is in bin structure. The gray jobs belong to tasks with period length q_1, the striped jobs to tasks with period length $q_2 = 3 \cdot q_1$, and the checkered jobs to tasks with period length $q_3 = 6 \cdot q_1$.

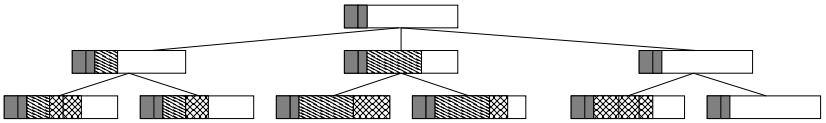

Figure 2. The bin tree corresponding to the schedule in Figure 1.

inverse is also true if no bin is overloaded. This allows us to model the basic PMP in terms of assignments of tasks to bins.

Lemma 2. *Let T be a set of tasks with period lengths $q_1 < \ldots < q_k$. Assume that for each task τ_i we are given a value $\ell_i \in \{0, \ldots, p_i/q_1 - 1\}$ (assigning the jobs of τ_i to bins $B_{\ell'}$ with $\ell_i \equiv \ell' \bmod p_i/q_1$). For each bin B_ℓ with $\ell \in \{0, \ldots, q_k/q_1 - 1\}$ denote by $T_\ell \subseteq T$ the tasks τ_i with $\ell_i \equiv \ell \bmod p_i/q_1$ (i.e., the tasks which run a job in B_ℓ). If for each bin B_ℓ we have that $\sum_{\tau_i \in T_\ell} c_i \leq q_1$, then there is a schedule for the tasks T on one machine. Moreover, this schedule can be found efficiently.*

Proof. From the assignment of tasks to bins the actual schedule can be computed by a greedy algorithm. Due to space constraints the full description and proof was moved to Appendix B. □

3 IP-Formulations

In what follows we describe several formulations for the basic PMP. First we consider a *time indexed formulation*, where we have variables assigning tasks to time slots on each machine. Then, we present a less naïve approach that exploits the algebraic feasibility criterion of Lemma 1. We call this model the *congruence-formulation*. It uses variables that indicate the offset of each task. We also describe a third model, the *bin-formulation*, that is specifically designed for the important case of harmonic periods. This last IP is based Lemma 2. It uses the concept of bins and directly assigns the tasks to the bins of the machines.

At the end of this section we explain how to model the additional constraints of the extended PMP. For sake of briefness, we define some global variables that are be used by all formulations. To this end, let $\mathscr{M}$ be a set of m identical machines. Since we are interested in minimizing the number of used machines, we assume that m is a precomputed upper bound on the total number of needed machines. For example, we can trivially take $m = |\mathscr{T}|$. In all our formulations we use variables $u_j \in \{0,1\}$ that are equal to one if machine $M_j \in \mathscr{M}$ is being used by some task. With these variables, the objective function is always to minimize $\sum_{M_j \in \mathscr{M}} u_j$.

3.1 Time-Indexed-Formulation

We first describe a naïve formulation that uses variables indicating whether a task starts processing at a certain time slot. Notice that this formulation has a pseudopolynomial number of variables. More precisely, we consider variables $w_{i,j,t} \in \{0,1\}$, which have a value of one if machine $M_j \in \mathscr{M}$ starts processing task $\tau_i \in \mathscr{T}$ at time t, and zero otherwise. For a task τ_i we only need these variables for $t \in \{0,\dots,p_i-1\}$, since for later time slots the task generates jobs periodically. We can enforce that all tasks are processed on some machine by requiring that for each task τ_i exactly one of the $w_{i,j,t}$ equals to one. To ensure that no two tasks collide, we employ variables $y_{i,j,t} \in \{0,1\}$ that equal one if and only if task τ_i is being processed on machine M_j during the time interval $[t,t+1)$. Again, we only introduce these variables for $t \in \{0,\dots,p_i-1\}$. The following restrictions link the $y_{i,j,t}$ variables to the $w_{i,j,t}$ variables:

$$\sum_{t'=t-c_i+1}^{t} w_{i,j,(t' \bmod p_i)} = y_{i,j,t} \qquad \forall t \in \{0,\dots,p_i-1\}, \forall \tau_i \in \mathscr{T}, \forall M_j \in \mathscr{M}. \tag{1}$$

Now we introduce linear constraints that prevent collisions. Two tasks $\tau_i, \tau_{i'}$ on machine M_j collide at time t if and only if $y_{i,j,(t \bmod p_i)} = y_{i,j,(t \bmod p_i)} = 1$. Due to the periodicity, it suffices to check timesteps t with $t \in \{0,\dots,\operatorname{lcm}(p_i,p_{i'})-1\}$ where $\operatorname{lcm}(p_i,p_{i'})$ denotes the least common multiple of p_i and $p_{i'}$. Therefore we can prevent tasks from colliding with the following set of restrictions:

$$y_{i,j,(t \bmod p_i)} + y_{i',j,(t \bmod p_{i'})} \leq 1 \quad \forall \tau_i, \tau_{i'} \in \mathscr{T}, \forall M_j \in \mathscr{M}, \forall t \in \{0,\dots,\operatorname{lcm}(p_i,p_{i'})-1\}. \tag{2}$$

These constraints then guarantee that our schedule has the required structure. We link the $u_j \in \{0,1\}$ with the $w_{i,j,t}$ variables by adding constraints $\sum_{t=0}^{p_i-1} w_{i,j,t} \leq u_j$ for all $\tau_i \in \mathscr{T}$ and for all $M_j \in \mathscr{M}$. A summary of the IP-formulation can be found in the appendix. The total number of variables is in $\Theta(|\mathscr{M}| \cdot |\mathscr{T}| \cdot q_k)$ and the number of constrains is in $\Theta(|\mathscr{T}|^2 \cdot |\mathscr{M}| \cdot q_k^2)$. Note that the due to Equality (1) the $y_{i,j,t}$ variables do not need to be included in the IP explicitly.

We also tried to decouple the assignment of the tasks to machines from the assignment of start offsets to tasks. However, even though this decreased the number of variables in our experiments the running times of the IP-solver increased.

3.2 Congruence-Formulation

Now we describe our congruence-formulation for the basic PMP. Its main concept is to introduce integer variables a_i which model the offset for each task. In order to check whether two tasks collide we derive linear constrains from the feasibility criterion given in Lemma 1.

For each task τ_i we introduce a variable $a_i \in \mathbb{N}$ which defines its offset. Additionally, we consider variables $x_{i,j} \in \{0,1\}$ that indicate, for each task $\tau_i \in \mathscr{T}$ and machine $M_j \in \mathscr{M}$, whether τ_i is assigned to M_j. To ensure that each task is actually assigned to a machine, we introduce the constraint $\sum_{M_j \in \mathscr{M}} x_{i,j} = 1$ for each task $\tau_i \in \mathscr{T}$.

It remains to ensure that no two tasks $\tau_i, \tau_{i'}$ on the same machine collide. Lemma 1 implies that it suffices to require that there is an integer $s_{i,i'}$ such that

$$c_{i'} \leq a_i - a_{i'} + s_{i,i'} \cdot \gcd(p_i, p_{i'}) \leq \gcd(p_i, p_{i'}) - c_i.$$

We want to enable the condition above only if two tasks τ_i and $\tau_{i'}$ share a machine. In order to achieve this we introduce variables $v_{i,i'}$ such that $v_{i,i'} = 1$ if τ_i and $\tau_{i'}$ are scheduled on the same machine. Hence, for each pair of tasks τ_i and $\tau_{i'}$ we introduce an integral variable $s_{i,i'}$ and the constraints

$$\begin{aligned} v_{i,i'} \cdot c_{i'} &\leq a_i - a_{i'} + s_{i,i'} \cdot \gcd(p_i, p_{i'}), \\ \gcd(p_i, p_{i'}) - c_i \cdot v_{i,i'} &\geq a_i - a_{i'} + s_{i,i'} \cdot \gcd(p_i, p_{i'}). \end{aligned}$$

Note that if $v_{i,i'} = 0$ there is always an integral value for $s_{i,i'}$ such that these constraints are satisfied (independently from the values for a_i and $a_{i'}$). For the variables $v_{i,i'}$ we add constraints of the form $v_{i,i'} \geq x_{i,j} + x_{i',j} - 1$ to ensure that they equal one if two tasks τ_i and $\tau_{i'}$ are scheduled on the same machine.

We summarize the congruence-formulation in the appendix. In total, we have $\Theta(|\mathscr{T}|^2 + |\mathscr{M}|)$ variables and $\Theta(|\mathscr{T}|^2 \cdot |\mathscr{M}|)$ constraints. The size of the formulation is thus polynomial in the input size.

3.3 Bin-Formulation

In this section we consider the case of harmonic period lengths. We make crucial use of the bin-tree concept explained in Section 2 to obtain a strong IP. The main idea is to define variables that model the assignment of tasks to bins on each machine. Then, we add restrictions to ensure that no bin is overloaded. By Lemma 2 this will imply a feasible schedule.

We first describe our model for the single machine case, where we only want to determine whether a set of tasks can be processed on one machine without collisions. In this case we already know the minimum period length of the tasks assigned to the machine, and thus the size of the bins is known to be q_1. We later generalize the model to the problem of minimizing the number of used machines.

Consider the single machine feasibility problem. Recall that in this case all bins have size q_1. We introduce a variable $z_{i,\ell} \in \{0,1\}$ that determines whether task $\tau_i \in \mathscr{T}$ is assigned to each bin B_ℓ, with $\ell \in \{0, \ldots, p_i/q_1 - 1\}$. Notice that we do not need to consider bins with $\ell \geq p_i/q_1$, since for these bins the schedule of τ_i is repeated periodically. First of all we require that all jobs are assigned to some bin by requiring $\sum_{\ell=0}^{p_i/q_1-1} z_{i,\ell} = 1$ for all $\tau_i \in \mathscr{T}$.

Now we ensure that no bin is overloaded. Notice that if τ_i is assigned to bin B_ℓ then this task creates jobs in all bins $B_{\ell'}$ so that $\ell' \equiv \ell \bmod p_1/q_1$. Equivalently, a job created by task τ_i is processed on bin B_ℓ if and only if $z_{i,(\ell \bmod p_i/q_1)} = 1$. Then, we can guarantee that a bin is not overloaded by imposing a knapsack type constraint

$$\sum_{\tau_i \in \mathscr{T}} c_i \cdot z_{i,(\ell \bmod p_i/q_1)} \leq q_1 \qquad \forall \ell \in \{0, \ldots, q_k/q_1 - 1\}.$$

In the multiple machine problem we have the extra difficulty that it is not known a priori which is the smallest period length appearing on each machine. Therefore, the size of the bins on a machine is not determined until all jobs are assigned to the machine.

As before, we consider variables $z_{i,j,\ell} \in \{0,1\}$ that indicates whether a task $\tau_i \in \mathscr{T}$ is assigned to machine $M_j \in \mathscr{M}$ on bin B_ℓ for $\ell \in \{0,\ldots,q_k/q_1-1\}$. We consider here that bin B_ℓ has size q_1. Note that in the case that machine M_j processes no task with period length q_1, we cannot really consider assignment of jobs to bins of size q_1, since some job may be partially assigned to more than one bin. We then must "glue" bins of size q_1 together to create new bins of size q_2 or larger. This can be describe mathematically by considering the sum of several variables $z_{i,j,\ell}$. If, for example, $q_2 = 2q_1$, then the variable describing whether a tasks τ_i is assigned to machine M_j in the first bin of size q_2 is equal to $z_{i,j,0} + z_{i,j,1}$.

We generalize the ideas just discussed by introducing dummy variables $z^r_{i,j,\ell} \in \{0,1\}$ that are equal to one if τ_i is assigned to machine i on the ℓ-th bin of size q_r, for $\ell \in \{0,\ldots,q_k/q_r-1\}$. Formally, the dummy variables are defined as follows

$$z^r_{i,j,\ell} = \sum_{\ell'=\ell\cdot q_r/q_1}^{(\ell+1)\cdot q_r/q_1-1} z_{i,j,\ell'} \qquad \forall \tau_i, \forall M_j, \forall r \in \{1,\ldots,k\}, \forall \ell \in \{0,\ldots,q_r/q_1-1\}.$$

In the case that there is a task with period length q_r, our assignment must satisfy that no bin of size q_r is overloaded. However, this should not be required for machines that have no job with period length q_r assigned to it. Therefore, we introduce variables $d_{j,r} \in \{0,1\}$ that equal one if there is a tasks with period q_r assigned to machine M_j.

$$d_{j,r} \geq \sum_{\ell=0}^{p_i/q_1-1} z_{i,j,\ell} \qquad \forall M_j \in \mathscr{M}, \forall r \in \{1,\ldots,k\}, \forall \tau_i : p_i = q_r$$

This yields the following inequalities for ensuring that no bin is overloaded.

$$\sum_{\tau_j \in \mathscr{T}} c_i \cdot z^r_{i,j,t} \leq q_r + (1-d_{j,r})q_k \quad \forall M_j \in \mathscr{M}, \forall \ell \in \{0,\ldots,q_k/q_r-1\}, \forall r \in \{1,\ldots,k\}$$

Finally, we must link the variables u_j to the variables $z_{i,j,\ell}$. This can be easily done with analogous constraints as in the time-indexed-formulation. Also note that the variables $x_{i,j}$ can be trivially introduced to our formulation. In the appendix we give a summary of our bin-formulation. In total, it needs $\Theta\left(|\mathscr{T}| \cdot |\mathscr{M}| \cdot \frac{q_k}{q_1}\right)$ variables and constraints.

3.4 Extended Constraints

In order to handle the conditions additionally introduced in the extended PMP we need to add more linear constraints. Due to space limitations we sketch them only briefly. For the memory restrictions we introduce knapsack constraints that model the limited memory on a machine. The constraints that some tasks have to be scheduled on different machines are modeled in a straight-forward manner by suitably linking the variables of the tasks. By introducing variables for the communication links on each machine, we ensure that a machine opens all links which are needed by its tasks. Further knapsack

constraints ensure that the total number of links and their total bandwidth does not exceed the resources on each machine. For each available machine we pre-define whether it is in the left or in the right cabinet. We introduce constraints which ensure that sets of tasks are distributed evenly on the cabinets if required.

4 Computational Results

In this section we present our computational results. We solved all real-world instances provided by Boeing within minutes, with all constraints of the extended PMP and some additional constraints which we describe later. The most difficult instance has 177 tasks and needs 16 machines. The period lengths are almost harmonic (see details below). Instances of this size are far beyond of what the time-indexed formulation and the congruence formulation can solve in a reasonable amount of time. However, the special design of the bin-formulation allowed to solve the instances within 15 minutes.

For benchmarking purposes we first analyze how our different models perform on random instances. Moreover, we study the quality of solutions obtained by a First-Fit heuristic. The heuristic orders the tasks by period length and execution time and greedily assigns them to the first machine where it can find a suitable start offset. Note that for the basic PMP in the harmonic case this is already a 2-approximation algorithm [EHN$^+$10]. In the non-harmonic case one can prove with similar arguments as in [EHN$^+$10] that the algorithm uses at most $2OPT + k - 1$ machines. Reflecting the theoretical results, our benchmarking shows that First-Fit has a good performance in the basic PMP. However, as we will see, it does not cope well with the additional constraints of the extended PMP.

Due to the novelty of our problem, there is no existing standard set of instances for benchmarking. Therefore, we must rely on generating random instances. We consider two ways of generating random instances: pure random instances and random perturbations of real-world instances arising at Boeing. We will call the latter instances the real-world perturbed (RWP) instances. There are four different settings: for the basic PMP, we consider the non-harmonic case with pure random instances, the harmonic case with pure random instances, and the harmonic case with RWP instances. For the extended PMP we benchmark only with RWP instances in the harmonic case.

All computations were done on a two-processor machine with Intel Xeon 2.66 GHz CPUs with 8 GB of RAM, running Linux. We used CPLEX release version 12.1.0.

We remark that additionally we introduce some cuts to the IP formulations. If for two tasks, the sum of their execution times exceeds the greatest common divisor of their periods, Lemma 1 implies that they cannot be assigned to the same machine. Thus we can add separation constraints for these tasks similar to those used in the extended PMP model. Moreover, for any assignment of tasks to a processor, the sum of their total required execution times during the hyperperiod may not exceed the hyperperiod. This is expressed with knapsack type constraints. Notice that in all our IP-formulations we need an upper bound on the number of machines. This was obtained by first running the First-Fit heuristic.

	IP-formulations				Heuristic
# tasks	CF		TIF		FF
10	0.11s	98%	–	0.00%	2.99%
20	2.52s	92%	–	0.00%	2.23%
30	277.41s	42%	–	0.00%	1.92%

Table 1. The table shows our computational results for the pure random instances in the non-harmonic case.

4.1 Non-harmonic Case

In the non-harmonic case we benchmark the following IP-formulations/algorithms: the time-indexed-formulation (TIF), the congruence-formulation (CF), and the First-Fit heuristic (FF). The pure random instances were created by taking five different period lengths uniformly at random from the set $\{2^x \cdot 3^y \cdot 50 \,|\, x \in \{0,\ldots,4\}, y \in \{0,\ldots,3\}\}$. This is a typical number of period lengths in real-world instances. For each task τ_i, its period length p_i is chosen uniformly at random from one of the five period lengths. (In our experiments we observed that larger values for the number of period lengths in an instance result in instances which are harder to solve; however, the relation of the running times between the three IP-formulations remains the same.) Its execution time is drawn from an inverse exponential distribution. This results in realistically small execution times in comparison with the period length and hence mimics the real instances from Boeing. We created 200 random instances each for the case of 10, 20, 30, 40, and 50 tasks. Whenever ten runs in a row did not finish before the timeout of 30 minutes or ran out of memory, we did not consider the respective formulation any further (denoted by dashes in the table).

Table 1 shows our computational results. In all our tables, for each IP-formulation the left column shows the average running times in seconds[4]. The right column shows the percentage of instances that could be solved to optimality within the time limit. For the First-Fit algorithm, we show the average relative error (in %) of the solutions with respect to the optimal solution. The running time of First-Fit is negligible.

Discussion. The First-Fit heuristic apparently performs very well, obtaining the optimal solution most of the time regardless of the number of tasks. This is somewhat surprising given that the problem is theoretically rather difficult (i.e., *NP*-hard to approximate within a factor of $|\mathscr{T}|^{1-\varepsilon}$), see [EHN$^+$10]. However, the instances created in that reduction are very special and not likely to arise in our random draws. We notice that TIF is impractical even for small instances due to the huge number of integer variables involved in the formulation. In comparison, CF does much better and is able to solve most instances with up to 30 tasks in reasonable time (less than 30 min.).

4.2 Harmonic Case

In the harmonic case we benchmark the following IP-formulations/algorithms: the time-indexed-formulation (TIF), the congruence-formulation (CF), the bin-formulation (BF)

[4] We use the shifted geometric mean of running times t_i calculated by $\left(\prod_{i=1}^{n}(t_i+1)\right)^{1/n}-1$. We use the shift in order to decrease the strong influence of the very easy instance in the mean values.

	IP-formulations						
# tasks	BF		CF		TIF		FF
10	0.28s (1×)	99%	0.25s (0.9×)	97%	–	0.00%	0.00%
20	1.8s (1×)	100%	6.54s (3.64×)	90%	–	0.00%	0.27%
30	8.2s (1×)	97%	369.39s (45.05×)	32%	–	0.00%	0.06%
40	36.64s (1×)	80%	–	0%	–	0.00%	0.70%

Table 2. Computational results for the pure random instances in the harmonic case (basic PMP).

	IP-formulations						Heuristic
# tasks	BF		CF		TIF		FF
10	0.01s (1×)	100.00%	0.35s (33.07×)	100.00%	2.79(265.54×)	98.49%	0.00%
20	0.19s (1×)	99.00%	32.51s (174.03×)	66.00%	260.3(1393.43×)	50.00%	1.26%
30	0.45s (1×)	98.50%	487.04s (1072.48×)	2.50%	–	0.00%	0.76%
40	1.17s (1×)	98.00%	–	0.00%	–	0.00%	1.36%
50	2.96s (1×)	97.50%	–	0.00%	–	0.00%	0.63%
60	7.25s (1×)	96.50%	–	0.00%	–	0.00%	0.85%
70	12.76s (1×)	95.00%	–	0.00%	–	0.00%	0.00%
80	28.47s (1×)	93.50%	–	0.00%	–	0.00%	0.09%
90	45.58s (1×)	89.00%	–	0.00%	–	0.00%	0.00%
100	113.89s (1×)	89.50%	–	0.00%	–	0.00%	0.00%
150	977.97s (1×)	74.42%	–	0.00%	–	0.00%	0.00%

Table 3. Computational results for the RWP instances (harmonic case) for the basic PMP.

and the First-Fit heuristic (FF). In the harmonic case the pure random instances were created by first generating a harmonic sequence of five periods in the following way: We start with period length 50 and successively generate the other periods by multiplying two, three, or six to the previous period. The periods and execution times for the tasks are drawn as in the non-harmonic case. The RWP instances were created by taking tasks uniformly at random from a large harmonic Boeing-instance and perturbing execution time and – for the extended PMP – the memory requirements randomly by up to 25 %. The other extended constraints remain unchanged.

For the pure random instance we consider the basic PMP only. When running the RWP instances we consider both the basic and the extended PMP. Tables 2, 3, and 4 show our computational results for the harmonic case. For the IP-formulations the value in parenthesis denotes the ratio between the respective running time and the time needed by the bin-formulation.

Discussion. In the three settings of the harmonic case the bin-formulation clearly outperforms the two other IP-formulations. While for small instances the congruence formulation is still competitive, as the number of tasks increases the bin formulation becomes superior. The time-indexed formulation failed to find an optimal solution before the timeout even on small instances with ten tasks.

This shows that taking the bin structure into account in the bin-formulation allows a significantly better running time in comparison with the other formulations. In contrast to the congruence formulation no modulo-operation has to be encoded in the IP-model

	IP-formulations						Heuristic
# tasks	BF		CF		TIF		FF
10	0.09s (1×)	100.00%	0.2s (2.37×)	100.00%	5.03(58.73×)	100.00%	4.02%
20	2.16s (1×)	98.50%	19.96s (9.23×)	84.50%	29.19(13.5×)	14.50%	15.15%
30	19.8s (1×)	99.00%	119.22s (6.02×)	20.00%	–	0.00%	27.81%
40	97.02s (1×)	93.00%	–	0.00%	–	0.00%	25.13%
50	401.75s (1×)	62.11%	–	0.00%	–	0.00%	28.40%
60	655.06s (1×)	30.00%	–	0.00%	–	0.00%	14.12%
70	644.54s (1×)	8.00%	–	0.00%	–	0.00%	33.33%

Table 4. Computational results for the RWP instances (harmonic case) for the extended PMP.

(recall the conditions for a collision derived in Lemma 1). Also, the number of variables is a lot smaller than in the time-indexed formulation.

The First-Fit heuristic performs very well for the basic PMP and finds an optimal solution for most instances. Even though theoretically First-Fit is only a 2-approximation algorithm, it performs much better in practice. However, for real-world data we need to consider the extended PMP. In these instances First-Fit mostly missed the optimum by a significant margin. Also, it cannot provide a certificate of optimality and hence, in real settings one has to resort to IP-formulations. Nevertheless, First-Fit can be used as a fast heuristic which computes an upper bound on the number of needed machines.

4.3 Original Boeing Instances

We solved each real-world instance from Boeing in less than 15 minutes to optimality. The most challenging one consists of 177 tasks, and an optimal solution uses 16 machines. The period lengths of tasks belong to the set $\{50, 100, 200, 400, 800, 1000, 2000\}$. Note that this instance is not harmonic. Nonetheless, the number of jobs having one of the problematic period lengths (that is, 1000 and 2000) were very small (three and six respectively). We transform the instance to be harmonic by taking the 3 tasks with period length 1000 and changing their periods to 200, and changing the 6 tasks with period 2000 to have period length 400. Note that a solution of the modified instance can be easily converted to a solution of the original instance. On the other hand, we could prove that the optimal solution of the restrictive instance is also optimal for the original instance since the separation constraints already contained a set of 16 tasks that had to be assigned to different machines.

The instances from Boeing also have an additional extra constraint not yet discussed: We are given subsets of tasks that must be processed on the same machine. We call these the *cohabitation constraints*. Moreover, for a subset of tasks, a predefined assignment of tasks to machines is already given as part of the input. We have not considered these constraints in the previous experiments for several reasons: For the RWP case it is not clear how to generate meaningful random perturbations of these constrains. Also, the First-Fit algorithms sometimes fail to produce feasible schedules, even though the respective instance has a solution. In particular combinations of cohabitation and cabinet constraints often require a more sophisticated approach than pure

greedy. Therefore, the IP formulations are much more appropriate for the real world instances.

References

[Bha98] R. Bhatia. *Approximation Algorithms for Scheduling Problems*. PhD thesis, University of Maryland, 1998.

[BHR93] S. K. Baruah, R. R. Howell, and L. E. Rosier. Feasibility problems for recurring tasks on one processor. In *Selected papers of the 15th International Symposium on Mathematical Foundations of Computer Science*, pages 3–20. Elsevier, 1993.

[BRTV90] S. Baruah, L. Rousier, I. Tulchinsky, and D. Varvel. The complexity of periodic maintenance. *Proceedings of the International Computer Symposium*, 1990.

[But04] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications*. Springer, 2004.

[CPPT07] T. G. Crainic, G. Perboli, M. Pezzuto, and R. Tadei. New bin packing fast lower bounds. *Computers & Operations Research*, 34:3439–3457, 2007.

[EHN+10] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese. Scheduling periodic tasks in a hard real-time environment. In *Proceedings of ICALP 2010*, Lecture Notes in Computer Science. Springer, 2010. To appear. Full paper at http://disopt.epfl.ch/webdav/site/disopt/shared/PM_EHNSVW10_report.pdf.

[FH02] K. Fleszar and K. S. Hindi. New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29:821–839, 2002.

[KAL96] J. Korst, E. Aarts, and J. K. Lenstra. Scheduling periodic tasks. *INFORMS Journal on Computing*, 8:428–435, 1996.

[KALW91] J. Korst, E. Aarts, J. K. Lenstra, and J. Wessels. Periodic multiprocessor scheduling. In E. H. L. Aaarts, J. van Leeuwen, and M. Rem, editors, *PARLE '91 Parallel Architectures and Languages Europe*, volume 505 of *Lecture Notes in Computer Science*, pages 166–178. 1991.

[Leu04] J. Y.-T. Leung. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. Chapman & Hall/CRC, 2004.

[LGW08] K.-H. Loh, B. Golden, and E. Wasil. Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers & Operations Research*, 35:2283–2291, 2008.

[MT90] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, revised edition, November 1990.

[SKJ97] A. Scholl, R. Klein, and C. Jürgens. BISON: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24:627–645, 1997.

[SL94] D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics*, 41:579–585, 1994.

[Van99] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86:565–594, 1999.

[VBJN94] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3:111–130, 1994.

[VdC99] J. M. Valério de Carvalho. Exact solution of bin packing problems using column generation and branch and bound. *Annals of Operations Research*, 86:629–659, 1999.

[WL83] W. D. Wei and C. L. Liu. On a periodic maintenance problem. *Operations Research Letters*, 2:90–93, 1983.

A Summaries of IP-Formulations

A.1 Time-Indexed Formulation

$$\min \sum_{j=1}^{m} u_j$$

$$\text{s.t.} \sum_{M_j \in \mathscr{M}} \sum_{t=0}^{p_i-1} w_{i,j,t} = 1 \qquad \forall \tau_i \in \mathscr{T}$$

$$\sum_{t'=t-c_j+1}^{t} w_{i,j,(t' \bmod p_i)} = y_{i,j,t} \qquad \forall t \in \{0,\dots,p_i-1\}, \forall \tau_j \in \mathscr{T}, \forall M_j \in \mathscr{M}$$

$$y_{i,j,t} + y_{i',j,t'} \leq 1 \qquad \forall \tau_i, \tau_{i'}, \forall M_j, \forall t \equiv t' \bmod \gcd(p_i, p_{i'})$$

$$y_{i,j,t} \leq u_j \qquad \forall \tau_i, \forall M_j, \forall t \in \{0,\dots,p_i-1\}$$

$$u_j \in \{0,1\} \qquad \forall M_j$$

$$w_{i,j,t} \in \{0,1\} \qquad \forall \tau_i, \forall M_j, \forall t \in \{0,\dots,p_i-1\}$$

$$y_{i,j,t} \in \{0,1\} \qquad \forall \tau_i, \forall M_j, \forall t \in \{0,\dots,p_i-1\}$$

A.2 Congruence Formulation

$$\min \sum_{M_j \in \mathscr{M}} u_j$$

$$\text{s.t.} \sum_{M_j \in \mathscr{M}} x_{i,j} = 1 \qquad \forall \tau_i \in \mathscr{T}$$

$$u_j \geq x_{i,j} \qquad \forall \tau_i \in \mathscr{T},\ \forall M_j \in \mathscr{M}$$

$$a_i - a_{i'} + s_{i,i'} \cdot \gcd(p_i, p_{i'}) \geq v_{i,i'} \cdot c_{i'} \qquad \forall \tau_i, \tau_{i'} \in \mathscr{T}$$

$$\gcd(p_i, p_{i'}) - c_i \cdot v_{i,i'} \geq a_i - a_{i'} + s_{i,i'} \cdot \gcd(p_i, p_{i'}) \qquad \forall \tau_i, \tau_{i'} \in \mathscr{T}$$

$$v_{i,i'} \geq x_{i,j} + x_{i',j} - 1 \qquad \forall \tau_i, \tau_{i'} \in \mathscr{T},\ \forall M_j \in \mathscr{M}$$

$$x_{i,j} \in \{0,1\} \qquad \forall \tau_i, \tau_{i'} \in \mathscr{T}$$

$$u_j \in \{0,1\} \qquad \forall M_j \in \mathscr{M}$$

$$s_{i,i'} \in \mathbb{Z} \qquad \forall \tau_i, \tau_{i'} \in \mathscr{T}$$

$$v_{i,i'} \in \{0,1\} \qquad \forall \tau_i, \tau_{i'} \in \mathscr{T}$$

$$a_i \in \mathbb{N}_0 \qquad \forall \tau_i \in \mathscr{T}$$

A.3 Bin-Formulation

In the following we show the summary of the Bin-formulation. We described the formulation considering the $z^r_{i,j,\ell}$ variables, but they can clearly be easily replaced in the model by the $z_{i,j,\ell}$ variables.

$$\min \sum_{M_j \in \mathscr{M}} u_j$$

$$\text{s.t.} \sum_{M_i \in \mathscr{M}} \sum_{\ell=0}^{p_i/q_1-1} z_{i,j,\ell} = 1 \qquad \forall \tau_i \in \mathscr{T}$$

$$\sum_{\ell'=\ell \cdot q_r/q_1}^{(\ell+1)\cdot q_r/q_1-1} z_{i,j,\ell'} = z^r_{i,j,\ell} \qquad \forall \tau_i \in \mathscr{T}, \forall M_j \in \mathscr{M}, \forall r \in \{1,\ldots,k\}, \forall \ell \in \{0,\ldots,q_r/q_1-1\}$$

$$\sum_{\tau_j \in \mathscr{T}} c_i \cdot z^r_{i,j,t} \leq q_r + (1-d_{j,r})q_k \qquad \forall M_j \in \mathscr{M}, \forall \ell \in \{0,\ldots,q_k/q_r-1\}, \forall r \in \{1,\ldots,k\}$$

$$\sum_{\ell=0}^{p_i/q_1-1} z_{i,j,\ell} \leq d_{j,r} \qquad \forall M_j \in \mathscr{M}, \forall r \in \{1,\ldots,k\}, \forall \tau_i : p_i = q_r$$

$$z_{i,j,\ell} \leq u_j \qquad \forall M_j \in \mathscr{M}, \forall \tau_i \in \mathscr{T}, \forall \ell = \{0,\ldots,q_k/q_1-1\}$$

$$z_{i,j,\ell} \in \{0,1\} \qquad \forall M_j \in \mathscr{M}, \forall \tau_i \in \mathscr{T}, \forall \ell = \{0,\ldots,q_k/q_1-1\}$$

$$z^r_{i,j,\ell} \in \{0,1\} \qquad \forall M_j \in \mathscr{M}, \forall \tau_i \in \mathscr{T}, \forall, \ell = \{0,\ldots,q_k/q_1-1\}, \forall r \in \{1,\ldots,k\}$$

$$d_{j,r} \in \{0,1\} \qquad \forall M_j \in \mathscr{M}, \forall r \in \{1,\ldots,k\}$$

$$u_j \in \{0,1\} \qquad \forall M_j \in \mathscr{M}$$

B Omitted Proof

Proof (of Lemma 2). We describe a greedy algorithm which constructs a schedule for the tasks T on one machine based on the assignments ℓ_i. We order the tasks ascendingly by period length. Assume for induction that all tasks with period length $q_{r'}$ with $q_{r'} < q_r$ have already been assigned. We assume that in each bin all jobs are ordered ascendingly by the period length of their respective tasks and with all idle time of the bin being at its end. Now consider the tasks with period length q_r. We consider the bins B_0 to B_{q_r/q_1}. For a bin B_ℓ we determine the set T_ℓ as defined in the lemma statement. We iterate over the tasks in T_ℓ with period length q_r. For each such task $\tau_i \in T_\ell$ we greedily assign τ_i the smallest start offset a_i such that $a_i \in B_{\ell_i} = [\ell_i \cdot q_1, (\ell_i+1) \cdot q_1)$ and τ_i does not collide with any previously assigned task. For each bin B_ℓ we have that $|B_\ell| = q_1 \geq \sum_{\tau_i \in T_\ell} c_i$ and only tasks in T_ℓ are assigned to B_ℓ. Since the idle time in B_ℓ is not fragmented we can always find a suitable value a_i. Moreover, this assignment sustains the ordering inside each bin stated above. Continuing this procedure for all tasks of all period lengths completes the construction of the schedule. The algorithm can clearly be implemented in polynomial time. □