FlexRay Schedule Optimization of the Static Segment

Martin Lukasiewycz, Michael Glaß, and Jürgen Teich
University of Erlangen-Nuremberg, Germany
{martin.lukasiewycz,glass,teich}@cs.fau.de

Paul Milbredt
I/EE-81, AUDI AG, Germany
paul.milbredt@audi.de

ABSTRACT

The FlexRay bus is the prospective automotive standard communication system. For the sake of a high flexibility, the protocol includes a static time-triggered and a dynamic event-triggered segment. This paper is dedicated to the scheduling of the static segment in compliance with the automotive-specific AUTOSAR standard. For the determination of an optimal schedule in terms of the number of used slots, a fast greedy heuristic as well as a complete approach based on Integer Linear Programming are presented. For this purpose, a scheme for the transformation of the scheduling problem into a bin packing problem is proposed. Moreover, a metric and optimization method for the extensibility of partially used slots is introduced. Finally, the provided experimental results give evidence of the benefits of the proposed methods. On a realistic case study, the proposed methods are capable of obtaining better results in a significantly smaller amount of time compared to a commercial tool. Additionally, the experimental results provide a case study on incremental scheduling, a scalability analysis, an exploration use case, and an additional test case to emphasis the robustness and flexibility of the proposed methods.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems

General Terms

Design, Algorithms

1. INTRODUCTION

State-of-the-art automotive networks consist of up to a hundred *Electronic Control Units* (ECUs), interconnected by several buses. Distributed applications are implemented on these ECUs with the goal to improve safety, comfort, and efficiency. Upcoming applications like *X-by-wire* systems that aim to replace mechanical or hydraulic systems require a higher bandwidth and increased accuracy compared to the predominant event-triggered *CAN* bus [4]. The *FlexRay* communication system [7] has been developed by an industrial consortium to address these issues. It offers a time-triggered and an event-triggered architecture, a bandwidth of 10 Mbit/s, and supports different topologies, i.e., linear bus, star and hybrid topologies. Thus, FlexRay is the prospective automotive standard communication system.

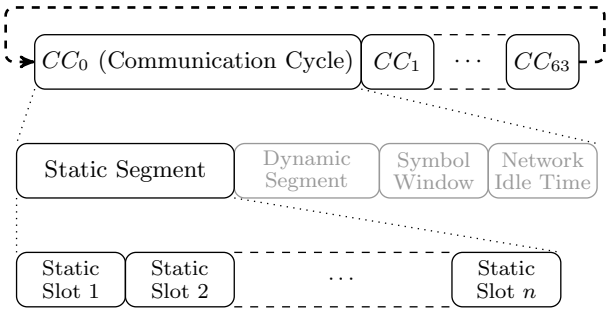


Figure 1: The FlexRay communication consisting of communication cycles with a detailed illustration of the static segment.

The FlexRay communication is organized in *cycles*, as illustrated in Fig. 1. Since each frame has exactly 6 cycle count bits, the cycles are numerated from 0 to 63 and subsequently start over with 0 again. Each cycle is divided into four segments of configurable duration:

(1) The *static segment* enabling a guaranteed real-time transmission of critical data,

(2) the *dynamic segment* (optional) for low-priority and event-triggered data,

(3) the *symbol window* (optional) used to transmit special symbols, and

(4) the *network idle time* used to perform a clock synchronization.

The focus of this paper is put on scheduling the static segment. The static segment is made up of n equally sized *slots* where each one is uniquely assigned to one *node* (or none). One node, however, may occupy more than one slot. Each slot consists of a header and trailer segment and a payload segment that is statically configured to carry between 0 and 254 bytes. By a predefined schedule, each slot is filled with the communication data of the applications, the *protocol data units* (PDUs). The PDUs are measured in bytes as the basic unit. In this paper, it is assumed that the basic communication units are PDUs, thus, the signals are already packed into PDUs. This is a common scenario, since PDUs are predefined by ECU and gateway packing strategies.

CODES+ISSS'09, October 11–16, 2009, Grenoble, France.

As illustrated in Fig. 1, the FlexRay protocol uses a time division multiple access (TDMA) to multiplex the communication into different slots. All participants of a FlexRay bus are aware of the current cycle number. As suggested in the AUTOSAR FlexRay Interface Specification [1], this value is used in series-production vehicles for a second multiplexing dimension to increase the utilization of the bus. Considering the AUTOSAR specification, the optimization of the schedule of the static segment becomes a challenging task that is the main subject of this paper.

The remainder of this paper is outlined as follows: Section 2 defines the problem of the schedule optimization of the static segment of a FlexRay bus and Section 3 presents related work on this topic. The schedule optimization is presented in Section 4, introducing a transformation from slot to bin packing and vice versa, a heuristic and complete ILP-based bin packing, as well as a metric and optimization method for the extensibility of a single slot. Section 5 presents the experimental results including a comparison to a commercial tool, a test case for incremental scheduling, and a scalability analysis of the proposed methods. Moreover, the introduced techniques are used for an exploration of the used slot size and one additional handmade test case is presented to study the exibility and robustness of the proposed methods. Finally, the paper is concluded in Section 6.

2. PROBLEM DEFINITION

Scheduling Requirements. In real-world implementations of the FlexRay bus, the periodic and safety-critical data is scheduled on the static time-triggered segment while the dynamic segment is mainly used for maintenance and diagnosis data [3, 20]. Though, in the first generation of the FlexRay bus in series-production vehicles, the static segment is not used at the full capacity [20], it is projected that the data volume on FlexRay buses will increase significantly in the future. Therefore, a schedule optimization that minimizes the number of used slots is necessary to (1) allow a high flexibility for incremental schedule changes[1] and (2) for future automotive networks with a higher data volume. Hence, an efficient schedule optimization of the static segment is the key to the success of the FlexRay bus.

The configuration of the FlexRay bus is defined by a large set of parameters. In particular, these parameters allow a configuration of the number and size of the slots in the static segment. Nevertheless, these values are mostly predefined by the manufacturer guided by existing data. For instance, the duration of a communication cycle is usually 5 milliseconds due to the periods of the PDUs in the present automotive networks that are predominantly a multiple of 5 milliseconds. For each PDU that is routed on the FlexRay bus, a fixed size in bytes is given and the minimal repetition is deduced from the period of the communication cycle and its own period, cf. [8]. In order to efficiently improve the tunable FlexRay parameters, fast scheduling techniques are necessary to allow for an effective parameter exploration.

AUTOSAR Interface Specification. As suggested in the AUTOSAR FlexRay Interface Specification [1] that is currently applied in all series-production vehicles, *cycle multiplexing* is used to increase the utilization of the FlexRay bus. An example of this *cycle multiplexing* for a single slot is illustrated in Fig. 2. The cycle multiplexing of PDUs is defined by the *base cycle* and the *cycle repetition*: The base cycle defines the offset in cycles for the first occurrence of the respective PDU. The cycle repetition denotes the frequency of a PDU in the multiplexing. The value of the cycle repetition is always a power of two $2^n, n \in \{0, ..., 6\}$ to allow a periodic occurrence in the 64 cycles. Thus, for a given base cycle b and repetition r, a PDU is existent in each communication cycle CC_i with

$$i = (b + r \cdot n)\%64 \text{ with } n \in \mathbb{N}_0.$$

An example of scheduling three PDUs m_0, m_1, and m_2 is given in Fig. 2. The base cycle values are 0 for m_0 and m_1 as well as 3 for m_2. The repetition values are 1 for m_0, 2 for m_1, and 4 for m_2. Given a common duration of a single communication cycle of 5 ms, the message m_0 is sent each cycle with a period of 5 ms, the message m_1 each second cycle with a period of 10 ms, and the message m_2 each fourth cycle with a period of 20 ms. The cycle multiplexing technique maximizes the utilization of the static segment in compliance with the high requirements for reliability and robustness and, therefore, is integrated into real-world automotive implementations of the FlexRay bus based on the AUTOSAR specification.

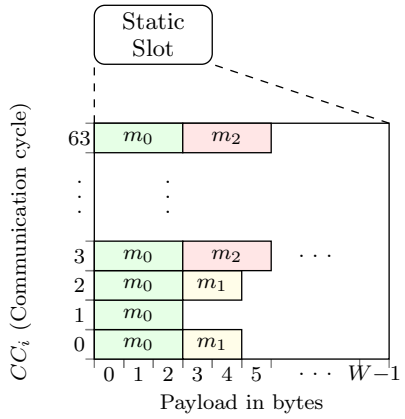


Figure 2: FlexRay cycle multiplexing of a single static slot. For a high utilization, the cycle multiplexing allows each slot to have an arbitrary message scheduling in each communication cycle.

3. RELATED WORK

The FlexRay specification [7] is under development by the *FlexRay Consortium* including *BMW*, *Daimler*, *General Motors*, and *Volkswagen*. Currently, the series-production vehicles using FlexRay are the *BMW X5*, *X6* and *7* series [2] and the *Audi A8* [12]. All these series-production vehicles are compliant with the FlexRay AUTOSAR Interface Specification [1]. Thus, this AUTOSAR specification is the defacto industrial standard for the software specification of the FlexRay nodes.

Recent papers cover diverse FlexRay related topics. An introduction of the FlexRay protocol and the operating mode in real-world automotive systems is presented in [1, 3, 23]. In [9, 19], a timing and performance analysis of FlexRay embedded systems is given, mostly focused on the dynamic segment. The determination and optimization of FlexRay schedules for the dynamic segment is discussed in [18]. The work in [13] presents a scheme for the acknowledgment and retransmission of data that is implemented on top of an existing FlexRay schedule in order to increase the reliability of FlexRay-based applications.

[1]Incremental changes are common in the automotive area to decrease the testing exposure.

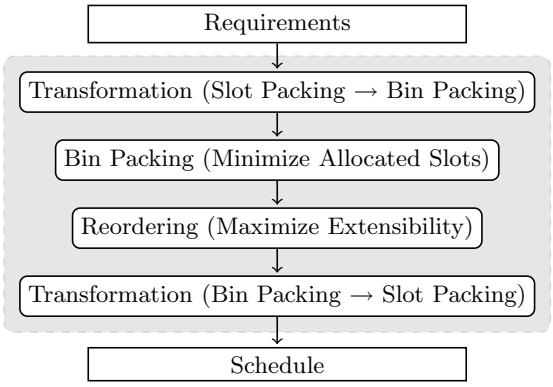


Figure 3: Schedule optimization flow.

An approach that optimizes the static segment with a *genetic algorithm* is proposed in [6]. The paper [24] introduces an ILP approach for a proposed custom software architecture. However, these papers do not consider the AUTOSAR [1] software specification for the FlexRay bus and are, therefore, not viable for current FlexRay scheduling problems in the automotive area. A recent work on scheduling the static segment is given in [8]. This paper includes the optimization of the schedule with respect to cycle multiplexing. However, a predefined frame packing as described in [21, 22] is assumed. This approach is rather restrictive since a two-level packing of signals to PDUs and PDUs to frames as provided by AUTOSAR is common in the automotive area. In contrast, the work at hand enables the scheduling of the PDUs directly into slots including the frame packing. Available tools for the scheduling problem as introduced in the previous section are TTX PLAN [25] based on an heuristic approach and DAVINCI NETWORK DESIGNER FLEXRAY [26], a graphical user interface that only allows to build schedules manually. The scheduling algorithm of TTX PLAN is not published, but the experimental results in this paper give evidence of an inferior behavior regarding runtime and quality of results.

One of the main contributions of this paper is a transformation scheme for the FlexRay scheduling problem into a special two-dimensional bin packing problem and its efficient solution. The general two-dimensional bin packing problem has been researched thoroughly, a brief summary is presented in [14]. Since an exact solution with an Integer Linear Program (ILP) results in a huge number of variables, cf. [5], heuristics are usually favored for unconstrained problems. However, in the presence of constraints like the level [15] or guillotine packing [20], an ILP can be formulated and solved efficiently. To the best of our knowledge, the special two-dimensional bin packing problem with individual level constraints, as presented in the work at hand, has not been topic of any research so far.

4. SCHEDULE OPTIMIZATION

The optimization flow as proposed in this paper is illustrated in Fig. 3. The main goal is to minimize the number of used slots in order to maximize the utilization of the bus. Unused slots are still part of the final schedule, but these slots can be assigned to any ECU if the schedule is extended incrementally in further development. Moreover, a fast scheduling approach is advantageous for, e.g., the exploration of specific bus parameters.

First, the transformation of the original slot packing problem into a special *bin packing* problem is performed using the proposed transformation scheme. The bin packing is carried out in order to minimize the number of allocated slots. This paper introduces a fast heuristic and a complete ILP approach for this special bin packing problem. Afterwards, the proposed *reordering* heuristic is used to further maximize the extensibility of the allocated slots by varying the position and base cycles of the PDUs in the slots. Finally, the transformation is inverted to convert the solution of the bin packing to a feasible FlexRay schedule. Since each slot is assigned to at most one ECU, the scheduling for each ECU is done independently and the slots are put together in the final schedule.

4.1 Problem Transformation

This section describes a one-to-one transformation between the slot packing problem that arises from the FlexRay cycle multiplexing and a special form of a two-dimensional bin packing problem. Since each slot corresponds to one bin, the transformation is presented for a single slot to a single bin and vice versa.

First, the general conditions for a feasible FlexRay slot packing are introduced: Each slot is defined by the payload size W (without the reserved load for the AUTOSAR specific update bits) and the number of cycles H which is 64 for the FlexRay bus. The set of PDUs is denoted M.

Each PDU $m \in M$ is defined by the following two values:

- $w_m \in \mathbb{N}$ - byte-length with $w_m \leq W$.
- $r_m \in \{2^n | n \in \{0, ..., 6\}\}$ - repetitions in the powers of two, defining the step-size for the multiplexing over the cycles. It holds that $r_m \leq H$.

For a feasible slot packing, two values for each PDU have to be determined:

- $x_m \in \mathbb{N}_0$ - the offset in bytes on the x-axis.
- $b_m \in \mathbb{N}_0$ - the base cycle that defines the offset on the y-axis. It holds that $b_m < r_m$.

A PDU is not allowed to exceed the slot ($x_m + w_m \leq W$) and no intersection between two PDUs is possible. The task of the transformation of the slot packing into a bin packing is to convert each PDU into a rectangle *element* and determine its position such that each feasible slot packing results in a feasible bin packing and vice versa. The bin size is the same as the slot size with the width W and height H. Also the position on the x-axis x_m and the width w_m for each element m correspond to the position and width of the PDU in the slot packing. Therefore, the main task is to find a transformation that obtains the following two values for each element m:

- $y_m \in \mathbb{N}_0$ - the offset on the y-axis.
- $h_m \in \mathbb{N}_0$ - the height of an element.

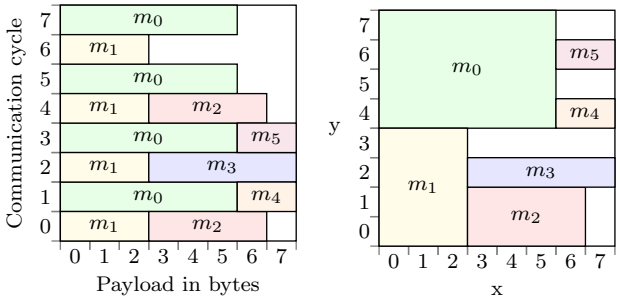
The transformation for the height h_m is related to the repetition r_m:

$$h_m = \frac{H}{r_m} \tag{1a}$$

$$r_m = \frac{H}{h_m} \tag{1b}$$

Thus, the height of an element equals the number of appearances of the corresponding PDU in the H cycles.

Given $b_m < r_m$, it follows that in the bin packing problem, the y_m position is restricted to r_m individual levels, depending on the height of the element. It holds that the *level* of an



(a) Slot (b) Bin

Figure 4: Example of a transformation from a slot packing problem (a) to a bin packing problem (b) and vice versa.

element $l_m = y_m/h_m$ has to be in $\mathbb{N}_0$. This arises from the fact that two PDUs with the same repetition but different base cycles will never intersect each other. The same holds for elements of the same height but different levels.

Consider the following transformation function $t : \mathbb{N}_0 \times \mathbb{N} \rightarrow \mathbb{N}_0$:

$$t(x,y) = \begin{cases} 0, & x = 0 \\ t(\frac{x}{2}, \frac{y}{2}), & x \text{ is even} \\ t(\frac{x-1}{2}, \frac{y}{2}) + \frac{y}{2}, & x \text{ is odd} \end{cases} \tag{2a}$$

with

$$y \in \{2^n | n \in \mathbb{N}_0\} \tag{2b}$$

$$0 \leq x < y \text{ and } x \in \mathbb{N}_0 \tag{2c}$$

It holds:

$$t(t(x,y),y) = x \text{ and } t(x,y) = t^{-1}(x,y) \tag{2d}$$

The transformation function t directly transforms the level of an element l_m to the base b_m and vice versa, such that the following holds:

$$l_m = t(b_m, r_m) \text{ and } b_m = t(l_m, r_m)$$

and, thus,

$$y_m = l_m \cdot h_m = t(b_m, r_m) \cdot \tfrac{H}{r_m} \tag{3a}$$

and

$$b_m = t(\tfrac{y_m}{h_m}, r_m) = t(\tfrac{y_m}{h_m}, \tfrac{H}{h_m}). \tag{3b}$$

Thus, a transformation from the slot packing problem to a bin packing problem with individual level constraints based on the height of the elements is done by applying Eq. (1a) and (3a) for each PDU. For the opposite direction, the transformation is performed by Eq. (1b) and(3b) for each element. An example for the transformation of a single slot is given in Fig. 4. A detailed proof for the correctness of this transformation is given in Appendix A.

4.2 Bin Packing

The task of a two-dimensional bin packing problem is to pack rectangular elements of different sizes defined by an individual width w and height h into a minimal number of rectangular bins without any intersection. Each bin has the fixed length W and height H. The transformation of the slot packing into a special two-dimensional bin packing problem determines a rectangular element m with the width w_m and height h_m for each PDU $m \in M$. The width of the bin equals the payload of a slot W and the height is the number of cycles which is 64 for FlexRay.

In contrast to the common two-dimensional bin packing, the transformed problem from the previous section contains two constraints:

(1) Each element $m \in M$ has a height h_m that is a power of two, i.e., 2^n with $n \in \mathbb{N}_0$ and the bin height is at least the maximal height of all elements.

(2) Each element $m \in M$ can be placed everywhere on the x-axis but only on a multiple of its height on the y-axis, i.e., $y_m = l \cdot h_m$ with the level $l \in \{0, ..., \frac{H}{h_m} - 1\}$.

This section introduces two optimization approaches for this specially constrained two-dimensional bin packing problem. The first approach is a fast greedy heuristic, the second is an efficient encoding as an ILP that allows to find the optimal solution.

4.2.1 Fast Greedy Heuristic

The presented bin packing problem can be solved by a fast greedy heuristic comparable to the approach presented in [20]. This heuristic is outlined in Alg. 1.

Algorithm 1 Fast greedy heuristic for bin packing.

```
1: S = {} //set of bins
2: for m ∈ M do
3:     for s ∈ S do
4:         if place(m, s) then
5:             continue with next m
6:         end if
7:     end for
8:     create new s and add it to S
9:     place(m, s)
10: end for
```

The algorithm starts with an empty set of bins S. Each element $m \in M$ is tried to be placed subsequently in a bin $s \in S$. Here, the function $place(m, s)$ is problem dependent and returns *true* if the placing is successful, and *false* otherwise. If an element is not placed in any of the allocated bins in S, a new bin s is allocated, added to S, and the element m is placed into this new empty bin.

Applied to the proposed special bin packing problem, the order of M influences the quality of the results. The elements in M are ordered first by their height h_m such that high elements are ordered to the front. The second criterion for the ordering of elements of the same height is the width w_m such that wide elements are ordered to the front. The function $place(m, s)$ tries to place each element m the most left void space in the bin s considering the individual level constraints of the proposed bin packing problem. This strategy tends to avoid the waste of void space of the bins. The complexity of this heuristic is polynomial.

4.2.2 ILP

Basic ILP. Solving a general two-dimensional bin packing problem with an Integer Linear Program (ILP) results in a high number of variables and constraints [5, 15]. However, the fact that each element has a height of a power of two and can only be placed on levels depending on its height can be exploited to deduce a compact and efficient ILP formulation with relatively few variables and constraints. The ILP formulation relies on the following binary variables:

- $\mathbf{m_{s,l}}$ - element m is placed at level l in bin s
- $\mathbf{s}$ - bin s is allocated (used)

The ILP is formulated as follows:

$$min \sum_{s \in S} \mathbf{s} \tag{4a}$$

$$\forall\, m \in M : \sum_{s \in S} \sum_{l=0}^{\frac{H}{h_m}-1} \mathbf{m_{s,l}} = 1 \tag{4b}$$

$$\forall\, s \in S, \{y = 0, ..., H-1\} : \sum_{m \in M} w_m \cdot \mathbf{m_{s,\lfloor \frac{y}{h_m} \rfloor}} \leq W \tag{4c}$$

$$\forall\, m \in M, s \in S, \{l = 0, ..., \tfrac{H}{h_m} - 1\} : \mathbf{s} - \mathbf{m_{s,l}} \geq 0 \tag{4d}$$

The objective function (4a) of the ILP minimizes the number of allocated bins. Here, the set S has to contain a minimal number of bins that are necessary to solve the problem. This number is deduced from the presented fast heuristic approach. The constraints (4b) state that each element m is placed in exactly one bin s at the specific level l. By adding the widths of the elements and restricting this sum by the width of a bin, the constraints (4c) ensure that the size of each bin is not exceeded. The constraints (4d) state that a bin s has to be allocated if at least one element m is placed in it.

Solving this ILP provides a bin s and level l for each element m (exactly one variable $\mathbf{m_{s,l}}$ is true). Placing the elements starting from the highest element to the most left void space in the bin s at the level l results in a feasible solution of the bin packing problem. This holds since the individual level constraints induce that each element that is sorted to the most left void space has at most one contact element on its left, cf. Fig. 4(b). Thus, the constraints (4c) are sufficient to determine a feasible bin packing.

Though this is a very efficient ILP encoding in terms of the number of variables, one has to keep in mind that the complexity of an ILP is exponential in general. Moreover, in contrast to the heuristic approach, the presented ILP cannot be used incrementally, i.e., an already allocated bin cannot be filled with additional elements without moving the old elements.

Enhanced ILP. The stated ILP can be further improved by reducing the search space by applying domain-specific knowledge. First, the set of S is reduced by one bin, simplifying the ILP by omitting several variables and constraints. In case there exists no feasible solution of this simplified ILP, there also exists no feasible bin packing for $|S|-1$ bins and, thus, the reference solution obtained by the heuristic is already optimal.

Furthermore, a lower bound for the objective is deduced by domain-specific knowledge to improve the runtime of the ILP. This lower bound is calculated as follows:

$$lb(M) = \frac{\sum_{m \in M} w_m \cdot h_m}{W \cdot H}. \tag{5a}$$

The additional constraint

$$\sum_{s \in S} \mathbf{s} \geq \lceil lb(M) \rceil \tag{5b}$$

sets the lower bound for the objective function. This constraint improves the runtime of the ILP: If the optimal solution is reached and equals the lower bound, the optimization process terminates immediately.

Regarding the problem of bin packing, a so-called *symmetry breaking* is applicable to reduce the search space. Consider the following one dimensional bin packing example: Given two elements m_1 and m_2 and two bins s_1 and s_2, there exist four possible distributions of the elements to the bins:

1. m_1, m_2 in s_1
2. m_1 in s_1 and m_2 in s_2
3. m_1, m_2 in s_2
4. m_1 in s_2 and m_2 in s_1

Since all bins have the same size and their order is negligible, there exists a symmetry between s_1 and s_2 regarding (1),(3) and (2),(4): Either (1) or (3) state that both elements are in the same bin, and correspondingly (2) or (4) state the both elements are in different bins. If the element m_2 is prohibited to be packed to bin s_2, (2) and (3) become invalid and the symmetry is broken. Thus, the search space is effectively reduced.

In order to generalize the symmetry breaking for the presented ILP formulation for the two-dimensional bin packing problem, two order functions for the elements and bins are used:

$$o: \quad S \rightarrow \mathbb{N} \tag{5c}$$

$$o: \quad M \rightarrow \mathbb{N} \tag{5d}$$

These functions assign to each element and bin, respectively, a unique integer value starting from 1 to $|M|$ and $|S|$, respectively. Given these functions, the symmetry breaking between bins is performed by adding the following constraints to the ILP formulation:

$$\forall m \in M, s, s' \in S(s \neq s'), l = \{0, ..., \tfrac{H}{h_m} - 1\}$$
$$\text{with } o(m) = o(s'), o(s') < o(s) : \mathbf{m_{s,l}} = 0 \tag{5e}$$

This ensures that an element is not allowed to be placed in a bin with a higher order.

Additionally, the two-dimensional bin packing leads to a possible horizontal symmetry through the middle of each bin. The symmetry breaking inside a bin is performed by adding the following constraints:

$$\forall m \in M, s \in S, l = \{0, ..., \left\lceil \tfrac{H}{2 \cdot h_m} - 1 \right\rceil\}$$
$$\text{with } o(m) = o(s) : \mathbf{m_{s,l}} = 0 \tag{5f}$$

This ensures that all elements with the same order value as the order value of a bin can only be placed in the lower half of this bin. Note that each symmetry breaking effectively accelerates the ILP solving without excluding optimal solutions.

4.2.3 Mutex Packing

In some cases, it becomes necessary to add mutual exclusion (*mutex*) to the bin packing, i.e., two elements are not allowed to be packed into the same bin. This requirement arises if the scheduling problem uses the *in-cycle repetition* of PDUs, cf. [2].

The in-cycle repetition allows the scheduling of PDUs with a lower period than the duration of one communication cycle. Consider the example in Fig. 5: Let the duration of a communication cycle t be $5ms$. Without the in-cycle repetition, the smallest available period is $5ms$ with the repetition $r = 1$. Given a PDU m_1 with the period $\frac{t}{2}$ ($2.5ms$), the in-cycle repetition allocates two slots with the approximate

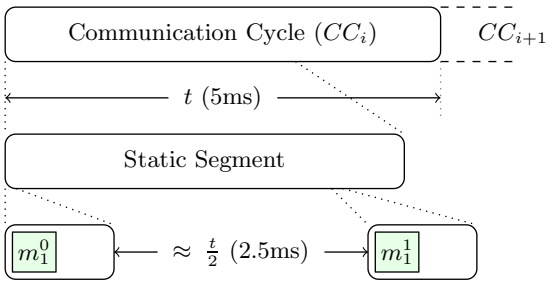


Figure 5: In-cycle repetition of a PDU m_1 in the static segment. The period of m_1 is twice the duration of the communication cycle by scheduling two instances of m_1 in two different appropriate slots.

distance of $\frac{t}{2}$ and schedules m_1 twice each communication cycle. Thus, for the PDU m_1, two instances m_1^0 and m_1^1 with the repetition $r = 1$ are scheduled. The requirement that both PDUs m_1^0 and m_1^1 are not scheduled in the same slot ensures that the slots can be arranged such that they have an approximate timing distance of $\frac{t}{2}$.

Extending the bin packing with the fast greedy heuristic from Section 4.2.1 by the mutex requirement is done by adding additional logic to the function $place(m, s)$. Given two elements m and $\tilde{m}$ that are prohibited to be placed in the same bin, the function $place(m, s)$ returns not only *false* if the void space in the bin is not enough for element m, but also if the element $\tilde{m}$ is already packed in the slot s. This has to be done for all pairs of mutually excluded elements.

In order to handle the mutex requirement in the ILP formulation given in Section 4.2.2, additional constraints become necessary. For each pair of mutex elements m and $\tilde{m}$, the following constraints are added to the formulation:

$$\forall s \in S, l \in \{0, ..., \tfrac{H}{h_m} - 1\} : -\mathbf{m_{s,l}} + \sum_{\substack{\tilde{s} \in S \setminus \{s\}, \\ \tilde{l} \in \{0,...,\frac{H}{h_{\tilde{m}}}-1\}}} \tilde{\mathbf{m}}_{\tilde{\mathbf{s}},\tilde{\mathbf{l}}} \geq 0 \tag{6}$$

In general, for the in-cycle repetition it holds $h_m = h_{\tilde{m}} = H$ resulting in $l = \tilde{l} = \{0\}$. These additional constraints restrict the search space, leading to an observed speedup of the ILP runtime.

4.3 Reordering for Extensibility

In real-world applications, an incremental scheduling is applied to allow scheduling new PDUs without changing the existing schedule, i.e., changing the position of the existing PDUs and slots. In order to minimize the number of allocated slots, the void space in partially used slots is exploited to schedule new PDUs. The *extensibility* of a slot describes its capability to allow scheduling additional PDUs.

This section introduces a metric for the extensibility of a bin, and, therefore, also for the corresponding slot. Moreover, a heuristic for the reordering of the elements in a bin is presented to maximize the extensibility value. The utilization of a bin is defined as:

$$U(s) = \sum_{m \in s} \frac{w_m}{W} \cdot \frac{h_m}{H} \tag{7a}$$

This value is between 0 and 1. In general, a lower utilization leads to a higher extensibility of a bin. However, the void space of a bin should be connected to enable the placement of large elements as well as several small elements. On the other hand, an unconnected void space hinders the placement of large elements. Note that large elements correspond to large or frequently sent PDUs. Thus, the goal is to reorder the elements in a bin in order to maximize a maximal rectangle over the void space. Fig. 6 illustrates two different element orderings in a bin. As a measure of extensibility, the following formula is proposed in the work at hand:

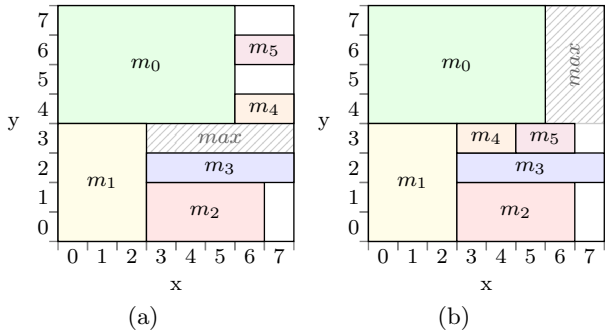


Figure 6: Extensibility metric with (a) a suboptimal solution of $E(s) = \frac{6}{64}$ and (b) the optimal solution of $E(s) = \frac{3}{64}$.

$$E(s) = 1 - U(s) - \frac{w_{max}}{W} \cdot \frac{h_{max}}{H} \tag{7b}$$

The values w_{max} and h_{max} are the width and height of the rectangle with maximal area over the void space. A small value indicates a high extensibility such that $E(s)$ should be minimized. Note that this formula has the advantage that inferior bin packings that produce a high number of bins are not favored since also bins with the utilization 1 have the optimal extensibility value 0.

Reordering the elements in a bin with the proposed metric is not possible with a complete method like an ILP since the objective function is not linear. Thus, an approach based on *Simulated Annealing* (SA) [11] is used. SA is a popular heuristic optimization approach that iteratively improves a single solution. At each step, the SA creates a *neighbor* solution based on the current solution and replaces it with a given probability or if the new solutions is better, respectively. The probability for the replacement of solutions s_x with s_y is calculated by

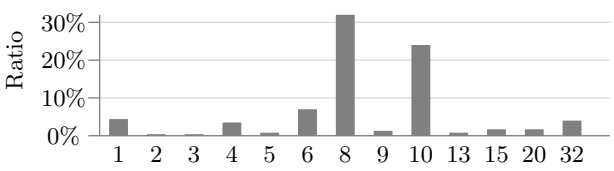
$$e^{-\frac{E(s_x)-E(s_y)}{\tau(i)}} \tag{7c}$$

where $\tau(i)$ is the *temperature* at iteration i. The temperature is calculated by a user defined continuously falling function $\tau(i)$ that simulates the annealing process. The optimization process is limited by a maximal number of iterations.

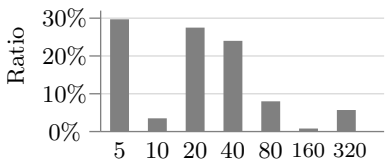
Searching for a *neighbor* of a bin packing cannot be performed straightforward since moving a single element randomly would create many infeasible solutions due to elements intersecting each other. Instead, the constraints defined in Eq. (4b) and (4c) are used in combination with the feasibility-preserving neighbor operator for linear constraint as proposed in [16]. This constrains the search space to the feasible bin packing solutions. Thus, a fast convergence to the optimal solution is reached.

5. EXPERIMENTAL RESULTS

A realistic example consisting of a FlexRay bus with 8 ECUs and an overall number of 220 PDUs is carried out as



(a) Distribution of the PDU sizes in bytes.



(b) Distribution of the PDU periods in milliseconds.

Figure 7: Distributions for PDU sizes and periods of the realistic case study.

a case study to show the applicability of the proposed methods. The distribution of the sizes and periods of the PDUs is illustrated in Fig. 7. The PDUs are highly heterogeneous in terms of their period and size. The parameters of the FlexRay bus are predefined such that the static segment consists of 62 slots with each slot carrying a payload of 42 bytes. Effectively, only 41 bytes are used since one byte is reserved for the update bits. The duration of the communication cycle is 5 milliseconds. The experiments were carried out on an Intel Pentium 4 3.20 GHz machine with 512 MB RAM. The ILP solver for the bin packing was the CPLEX solver in the version 10.5 [10]. The reordering was performed with the heuristic optimization framework OPT4J in the version 1.3 [17]. Here, the number of iterations for the simulated annealing was set to $n = 1500$ and the temperature was calculated by the following function:

$$\tau(i) = 10^{-(1+2\frac{i}{n})}$$

As described in the related work in Section 3, to the best of our knowledge, there exists no publication regarding the FlexRay bus scheduling in compliance with the industrial AUTOSAR Interface Specification [1] as presented in the work at hand. Currently, the only available automatic scheduling approach compliant with the AUTOSAR specification is the commercial tool TTX PLAN [25]. For the introduced case study, a reference solution obtained by TTX PLAN is available. Thus, a comparison is given in the following subsection. Since the tool TTX PLAN is not available for evaluation, no reference solutions are available for the subsequent experimental results.

5.1 Schedule Optimization

For all test cases, the time for the transformation into a bin packing problem and vice versa was omitted since it is negligibly small (less than 1 millisecond). The heuristic and ILP bin packing were tested, both with an additional reordering. By using the reordering strategy, the calculated average extensibility value $E(s)$ for all slots was minimized. The results for the case study are given in Tab. 1. The runtimes for the heuristic and ILP approach for the case study is a fraction of a second. The row ILP* in Table 1 shows the results that were obtained by the ILP approach without the presented enhancements. Here, the runtime is significantly higher with $25.7s$. In fact, the ILP enhancements are always advantageous and are used in the following for all experiments since there arises no additional overhead. The heuristic and ILP approach both deliver solutions with 27 allocated slots. Moreover, the ILP approach proves that 27 allocated slots is the optimal solution. The presented reordering heuristic improves the average extensibility value $E(s)$ significantly in approximately 45 seconds for both methods.

Method	runtime	slots	avg. $E(s)$
Heuristic	**0.065s**	**27**	0.029
+ Reordering	+42.5s	**27**	0.016
ILP* (CPLEX)	25.7s	**27**	0.028
ILP (CPLEX)	0.080s	**27**	0.028
+ Reordering	+48.0s	**27**	**0.019**
TTX Plan	360s	29	0.043

Table 1: Results for the case study.

The proposed algorithms were compared to a result obtained by the commercial available scheduling tool TTX PLAN [25] that is based on an undisclosed heuristic approach with a prospected polynomial scalability of the runtime. While the commercial tool returns a schedule with 29 allocated slots, the heuristic and the ILP improve this value by two slots, which is significant for a real-world application. These results show that the commercial tool delivers an inferior result in a comparatively large amount of time, in particular, in 6 minutes. The runtime of the commercial tool TTX PLAN and the presented approaches differs by four orders of magnitude. Since scheduling is typically just one of several tasks in a complete design flow, this enables design space explorations with reasonable runtimes using the proposed algorithms.

5.2 Incremental Scheduling

In contrast to the ILP approach, the heuristic scheduling method allows an *incremental scheduling*, i.e., an existing schedule is extended by additional PDUs without changing the position of the existing slots and PDUs. The incremental scheduling cannot guarantee an optimal schedule, but has the advantage that integration tests, as required in safety critical automotive applications, are kept at minimum. Thus, an incremental scheduling might be favored if the number of allocated slots is still not critical since integration tests are time-consuming and expensive.

In order to test the ability of the heuristic approach to extend an existing schedule incrementally, 60 random PDUs are generated following the distribution in Fig. 7. These 60 PDUs are distributed randomly among the 8 ECUs. The schedule generated in the previous subsection with the ILP approach is used as the base schedule that shall be extended. The results are given in Table 2. Starting from an existing schedule with 27 allocated slots and an extensibility value ($E(s)$) of 0.028 as determined by the ILP approach without additional reordering of the PDUs, 11 additional slots are necessary to schedule the new PDUs. In contrast, starting from the same schedule with an extensibility ($E(s)$) value of 0.019 as improved by the reordering, the necessary slots are 10 decreased by one slot compared to the approach without reordering. Thus, the reordering and the lower extensibility value show the benefit for this example of incremental scheduling. The runtime of the incremental scheduling is fairly low with 20 or 22 milliseconds, respectively. For comparison only, Table 2 shows the results of the scheduling of all 280 PDUs at once using the ILP approach, without incrementally changing an existing schedule. In this case, the

Method	runtime	slots	avg. $E(s)$
ILP (0.028) → Heuristic	0.022s	38	0.036
ILP (0.019) → Heuristic	0.020s	**37**	0.035
ILP (CPLEX)	12.2s	35	0.033

Table 2: Results of the incremental scheduling for the given case study.

optimal schedule is determined in 12.2 seconds with 35 slots. This also shows that the incremental scheduling is relatively efficient with only two additional slots if the reordering is applied.

Adding additional ECUs to an existing FlexRay bus system without additional PDUs for existing ECUs is not considered as incremental scheduling. In this case, the ILP or heuristic approach can be used separately for the new ECUs without any restriction.

5.3 Scalability Analysis

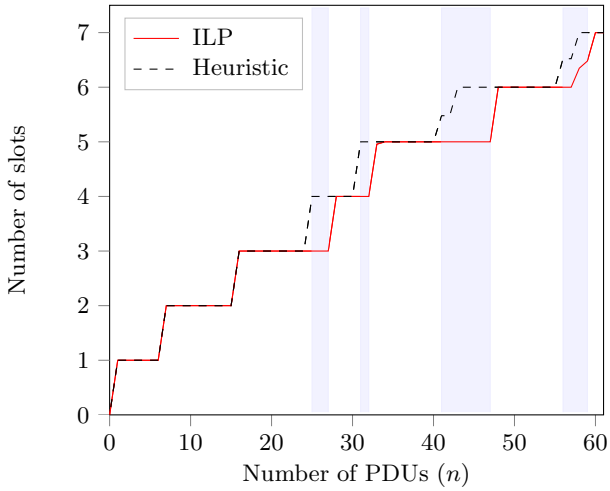
As a study for the scalability of the presented approaches, a scheduling for a single node is performed. An ordered list of 60 random PDUs is generated from the distribution of the given case study in Fig. 7. In this experiment, the first n PDUs of the ordered list are scheduled ensuring a monotonicity of the results. The slot size is 41 bytes as in the given case study. The study is performed for the ILP and heuristic approach using the average of 100 runs. For the ILP approach, a timeout of 120s is used.

The results are given in Fig. 8. Figure 8(a) shows the number of allocated slots per number of PDUs, and Fig. 8(b) shows the corresponding runtime for the algorithms. The number of allocated slots is always better for the ILP. On the other hand, the heuristic performs well with only a single additional necessary slot for some n values. The runtime of the heuristic approach shows a polynomial character as prospected (the plot in Fig. 8(b) is given in log-scale). In contrast, the runtime of the ILP approach highly depends on the current problem instance. If the heuristic finds an optimal result, the lower bound for the objective from Equation (5b) is trivially unsatisfied for $|S|-1$ slots and the runtime of the ILP approach equals the runtime of the heuristic. The runtime of the ILP is significantly higher if the current set of PDUs results in a number of allocated slots that allows an improvement compared to the heuristic approach.

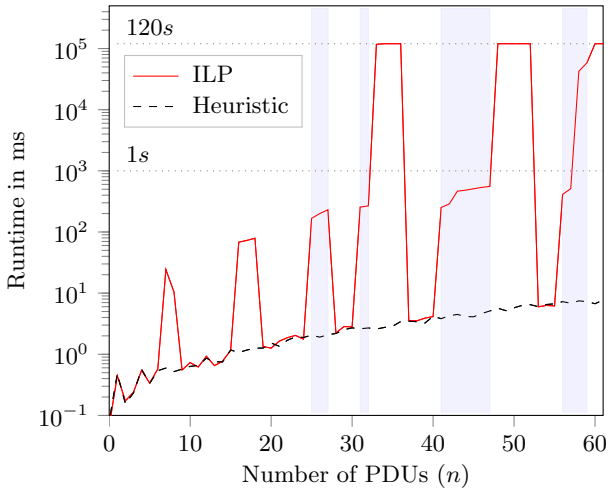
The results show that the ILP could improve the number of allocated slots in some cases by one slot within approximately one second. On the other hand, the ILP approach did timeout for some cases not returning a better result than the heuristic. Thus, for real-world applications, an appropriate timeout should be chosen for the ILP approach to enable a reasonable runtime. Note that scaling the number of the ECUs has a linear complexity since the bin packing and reordering is done separately for each ECU.

5.4 Slot Size Exploration

As a use case for the requirement for fast and optimal scheduling, the ILP approach is used for an exhaustive exploration of the payload size for the static slots. This optimization results in many trade-off solutions, since one objective is to maximize the cumulative payload of the unused slots for a higher extensibility and another objective is to maximize the number of unused slots for a higher flexibility. The number of slots is adapted such that the duration of the static segment is approximately 4.03 ms. The FlexRay protocol allows a configuration of the payload from 0 to 254



(a) Number of allocated slots depending on the number of scheduled PDUs.



(b) Runtime of the scheduling approaches depending on the number of scheduled PDUs.

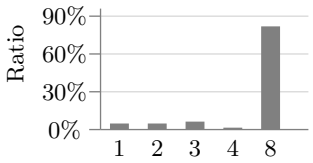
Figure 8: The result of the scalability analysis.

bytes whereas this value has to be a multiple of two. The exploration was performed from 34 bytes payload per slot as the minimal value due to the maximal message length and one additional byte for the update bits to 120 bytes. The runtime of the exploration was 435 seconds. The results of the cumulative payload of the unused slots depending on the slot payload size is given in Fig. 9. Obviously, a larger payload size of the static slots leads to less static slots per cycle and, thus, less communication overhead and in general to a higher cumulative payload of the unused slots. Compared to the predefined value of 42 bytes (35 unused slots), the solutions with 56 bytes (29 unused slots), 66 bytes (26 unused slots), and 76 bytes (24 unused slots) are considerable trade-off solutions. Based on these results and the knowledge that all values are optimal, a designer can choose the appropriate parameter for the slot size.

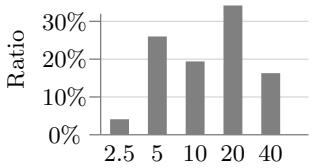
5.5 Supportive Test Case

In order to show the flexibility of the proposed approach, a supportive case study is used. This handmade case study is based on the FlexRay configuration and scheduling in the

Figure 9: Comparison of the payload per slot and the optimal cumulative payload of unused slots determined by the ILP approach.



(a) Distribution of the PDU sizes in bytes.



(b) Distribution of the PDU periods in milliseconds.

Figure 10: Distributions for PDU sizes and periods of the supportive case study.

BMW series 7 as provided in [2]: Overall 15 nodes are interconnected by a FlexRay bus. The duration of the communication cycle is the automotive typical value of $5ms$ with $3ms$ for the static segment and $2ms$ for the dynamic segment. The static segment consists of 91 slots each having a payload of 16 bytes. The periods of the PDUs, that are scheduled on the static segment, are $5ms$, $10ms$, $20ms$, and $40ms$. Moreover, approximately 10 PDUs with the period of $2.5ms$ are scheduled by a double in-cycle repetition as shown in the Section 4.2.3 and Fig. 5. The maximal size of the PDU is 8 bytes. The current schedule of the *BMW series 7* uses approximately two-third of the static slots.

Using the given information, 237 random PDUs were generated from the distribution in Fig. 10 and assigned randomly to 15 ECUs. Compared to the first case study, this handmade test case is more homogeneous in the size and period of the PDUs. The results of scheduling using the heuristic and ILP approach are given in Table. 3. The heuristic finds a schedule with 65 used slots in $12ms$, the extensibility value $E(s)$ is decreased from 0.028 to 0.011 in $5.3s$. Compared to this, the ILP approach finds the optimal schedule with 63 slots improving the heuristic approach by two slots. The runtime of the ILP is fairly low with $1.2s$ (the ILP* without the presented enhancements needs $2.7s$). The extensibility value $E(s)$ is comparable to the heuristic approach with 0.026 and the reduction to 0.011 in $4.9s$.

This supportive case study shows the robustness and flexibility of the presented scheduling approaches. The ILP and heuristic approach perform well in runtime and quality of results also on this fundamentally different test case.

Method	runtime	slots	avg. $E(s)$
Heuristic	0.012s	65	0.028
+ Reordering	+5.3s	65	**0.011**
ILP* (CPLEX)	2.7s	**63**	0.026
ILP (CPLEX)	1.2s	**63**	0.026
+ Reordering	+4.9s	**63**	**0.011**

Table 3: Results for the supportive case study.

6. CONCLUSION

This paper presents a scheduling optimization scheme for the static segment of the FlexRay bus in compliance with the AUTOSAR specification. First, the problem is transformed into a special two-dimensional bin packing problem using a proposed one-to-one transformation scheme. This constrained bin packing problem is solved either with a presented heuristic approach delivering good results in a relatively small amount of time or an introduced efficient ILP approach that delivers the optimal solution. Moreover, the paper presents a metric for the extensibility of an allocated slot and a heuristic based on Simulated Annealing is proposed to improve a schedule concerning this value.

The results of the realistic case study show that the heuristic and ILP approach are superior to a commercial tool in runtime and quality. An incremental scheduling shows that the presented extensibility metric is sound, being capable of saving additional slots. The scalability analysis studies the applicability of the proposed methods. For the fast and optimal scheduling, a use case is given by an exploration for the optimal payload size of the slots for the static segment. Finally, a supportive case study shows the flexibility and robustness of the proposed algorithms.

7. REFERENCES

[1] AUTOSAR. Specification of the FlexRay Interface Version 3.0.2, 2008. http://www.autosar.org.

[2] J. Berwanger, M. Peteratzinger, and A. Schedl. FlexRay startet durch - FlexRay-Bordnetz für Fahrdynamik und Fahrerassistenzsysteme (in German). In *Elektronik automotive: Sonderausgabe 7er BMW*, 2008. Available at http://www.elektroniknet.de/home/automotive/bmw-7/flexray-startet-durch/.

[3] J. Broy and K. D. Müller-Glaser. The impact of time-triggered communication in automotive embedded systems. In *Proc. of the SIES '07*, pages 353–356, 2007.

[4] CAN. Controller Area Network. http://www.can.bosch.com/.

[5] N. Christofides and E. Hadjiconstantinou. An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts. *European Journal of Operational Research*, 83(1):21–38, 1995.

[6] S. Ding, N. Murakami, H. Tomiyama, and H. Takada. A ga-based scheduling method for FlexRay systems. In *Proc. of the EMSOFT '05*, pages 110–113, 2005.

[7] FlexRay Consortium. FlexRay communications systems - protocol specification version 2.1 rev. a., 2005. http://www.flexray.com.

[8] M. Grenier, L. Havet, and N. Navet. Configuring the communication on FlexRay: the case of the static segment. In *Proc. of the ERTS '08*, 2008.

[9] A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P. Sampath, P. V. V. Ganesan, and S. Ramesh. Performance analysis of FlexRay-based ECU networks. In *Proc. of the DAC '07*, pages 284–289, 2007.

[10] ILOG. CPLEX. http://www.ilog.com/products/cplex/, Version 10.5.

[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220, 4598:671–680, 1983.

[12] J. Kötz and S. Poledna. Making FlexRay a Reality in a Premium Car. In *Proc. of the SAE International '08*, 2008.

[13] W. Li, M. Di Natale, W. Zheng, P. Giusto, A. Sangiovanni-Vincentelli, and S. Seshia. Optimizations of an Application-Level Protocol for Enhanced Dependability in FlexRay. In *Proc. of the DATE '09*, 2009.
[14] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1-3):379–396, 2002.
[15] A. Lodi, S. Martello, and D. Vigo. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, 8(3):363–379, 2004.
[16] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich. A feasibility-preserving local search operator for constrained discrete optimization problems. In *Proc. of the CEC '08*, pages 1968–1975, 2008.
[17] Opt4J. Meta-heuristic Optimization Framework for Java. http://www.opt4j.org/.
[18] T. Pop, P. Pop, P. Eles, and Z. Peng. Bus access optimisation for FlexRay-based distributed embedded systems. In *Proc. of the DATE '07*, pages 51–56, 2007.
[19] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the FlexRay communication protocol. In *Proc. of the ECRTS '06*, pages 203–216, 2006.
[20] J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 127(3):1304–1327, 2007.
[21] R. Saket and N. Navet. Frame packing algorithms for automotive applications. *Journal Embedded Computing*, 2(1):93–102, 2006.
[22] K. Sandstrom, C. Norstom, and M. Ahlmark. Frame packing in real-time communication. In *Proc. of the RTCSA '00*, pages 399–403, 2000.
[23] A. Schedl. Goals and architecture of FlexRay at BMW. In *Slides presented at the Vector FlexRay Symposium*, 2007. Available at https://www.vector-worldwide.com/.
[24] K. Schmidt and E. G. Schmidt. Message scheduling for the FlexRay protocol: The static segment. *IEEE Transactions on Vehicular Technology*, 58(5):2170–2179, 2009.
[25] TTTech. TTX Plan. http://www.tttech-automotive.de/.
[26] Vector. DaVinci Network Designer FlexRay. http://www.vector.com/.

APPENDIX

A. PROOF OF PROBLEM TRANSFORMATION

In the following, a proof for the problem transformation from Section 4.1 is given.

THEOREM 1. *Two elements in the generated bin packing intersect if and only if the corresponding PDUs in the slot packing problem are conflicting, i.e., intersecting.*

PROOF. For the x-transformation this is trivial since the x-position and width w of the PDUs or elements, respectively, are identical. Thus, this has to be solved for the y-transformation as given by the Equations (1a), (1b), (3a), and (3b).

The function $t(x, y)$ from Eq. (2a) performs a bitwise flip operation of the value x given in *Little-endian* encoding on its $\log_2 y$ bits. This satisfies the properties of function t in Eq. (2d) under the assumptions in Eq. (2b) and (2c).

Two elements in a bin intersect if

$$\neg((y_m + h_m \leq y_{\tilde{m}}) \vee (y_m \geq y_{\tilde{m}} + h_{\tilde{m}})) \tag{8a}$$

or by applying *De Morgan's law*

$$(y_m + h_m > y_{\tilde{m}}) \wedge (y_m < y_{\tilde{m}} + h_{\tilde{m}}). \tag{8b}$$

Applying Eq. (1a) to Eq. (8b) results in

$$y_m - y_{\tilde{m}} > -\frac{H}{r_m} \text{ and } y_m - y_{\tilde{m}} > \frac{H}{r_{\tilde{m}}}. \tag{8c}$$

A further transformation with Eq. (3a) results in the following equations:

$$r_m \cdot t(b_{\tilde{m}}, r_{\tilde{m}}) < r_{\tilde{m}} \cdot (t(b_m, r_m) + 1) \tag{8d}$$
$$r_{\tilde{m}} \cdot t(b_m, r_m) < r_m \cdot (t(b_{\tilde{m}}, r_{\tilde{m}}) + 1) \tag{8e}$$

If Eq. (8d) and (8e) are both satisfied, the two elements are intersecting. On the other hand, if either Eq. (8d) or (8e) is violated, the two elements are not intersecting.

($\Rightarrow$) If the slot packing for two PDUs m and $\tilde{m}$ is conflicting, the corresponding elements in the bin packing intersect:

Without loss of generality, it is assumed that $r_m \leq r_{\tilde{m}}$. Two PDUs are conflicting in the slot packing if

$$\exists n \in \mathbb{N}_0 : b_m + r_m \cdot n = b_{\tilde{m}}. \tag{8f}$$

This means, that the $\log_2 r_m$ least significant bits of b_m and $b_{\tilde{m}}$ are equal and, thus,

$$t(b_{\tilde{m}}, r_{\tilde{m}}) = \frac{r_{\tilde{m}}}{r_m}(t(b_m, r_m) + a) \tag{8g}$$

holds with

$$0 \leq a \leq 1 - \frac{r_m}{r_{\tilde{m}}} < 1. \tag{8h}$$

Here, a is the potential remainder by applying a shift of $\log_2 \frac{r_{\tilde{m}}}{r_m}$ bits. The upper bound 1 holds due to the problem-specific constraint $r_m \geq 1$ and the given assumption $r_{\tilde{m}} \geq r_m$.

Equation (8g) is transformed to the following two equations:

$$r_m \cdot t(b_{\tilde{m}}, r_{\tilde{m}}) = r_{\tilde{m}} \cdot (t(b_m, r_m) + a) \tag{8i}$$

$$r_{\tilde{m}} \cdot t(b_m, r_m) = r_m \cdot (t(b_{\tilde{m}}, r_{\tilde{m}}) - a \cdot \tfrac{r_{\tilde{m}}}{r_m}) \tag{8j}$$

Given Eq. (8i) with $a < 1$, Eq. (8d) holds. At the same time, Eq. (8e) holds due to Eq. (8j) and $a \geq 0$. Thus, the elements intersect as required.

($\Leftarrow$) If the slot packing for two PDUs m and $\tilde{m}$ is not conflicting, the corresponding elements in the bin packing do not intersect:

Without loss of generality, it is assumed that $r_m \leq r_{\tilde{m}}$. Two PDUs are not conflicting in the slot packing if

$$\forall n \in \mathbb{N}_0 : b_m + r_m \cdot n \neq b_{\tilde{m}}. \tag{8k}$$

This means, that the $\log_2 r_m$ least significant bits of b_m and $b_{\tilde{m}}$ are not equal and, thus, either

$$t(b_{\tilde{m}}, r_{\tilde{m}}) \leq \frac{r_{\tilde{m}}}{r_m}(t(b_m, r_m) - 1 + a) \tag{8l}$$

or

$$t(b_{\tilde{m}}, r_{\tilde{m}}) \geq \frac{r_{\tilde{m}}}{r_m}(t(b_m, r_m) + 1 + a) \tag{8m}$$

hold, both with a in the bounds from Eq. (8h). From Eq. (8l) it follows

$$r_{\tilde{m}} \cdot t(b_m, r_m) \geq r_m \cdot (\tfrac{r_{\tilde{m}}}{r_m}(1 - a) + t(b_{\tilde{m}}, r_{\tilde{m}})) \tag{8n}$$

that violates Eq. (8e) due to $\frac{r_{\tilde{m}}}{r_m}(1 - a) \geq 1$ that holds since $a \leq 1 - \frac{r_m}{r_{\tilde{m}}}$ as stated in Eq. (8h). Equation (8m) equals

$$r_m \cdot t(b_{\tilde{m}}, r_{\tilde{m}}) \geq r_{\tilde{m}} \cdot (t(b_m, r_m) + 1 + a) \tag{8o}$$

that violates Eq. (8d) due to $a \geq 0$. Thus, either Eq. (8d) or (8e) is violated and the elements do not intersect as required.

This proves Eq. (1a) and Eq. (3a). Equation (1b) holds due to Eq. (1a) and Eq. (3b) holds due to Eq. (3a) with the inverse properties of the t function given in Eq. (2d).

□