How to minimize cycle times of robot manufacturing systems

Tobias Hofmann¹ · David Wenzel^{2,3}

Received: 31 October 2019 / Revised: 17 April 2020 / Accepted: 28 June 2020 / Published online: 18 July 2020
© The Author(s) 2020

Abstract

The employment of industrial robot systems especially in the automotive industry noticeably changed the view of production plants and led to a tremendous increase in productivity. Nonetheless, rising technological complexity, the parallelization of production processes, as well as the crucial need for respecting specific safety issues pose new challenges for man and machine. Our goal is to develop algorithms, guidelines, and tools that make the commissioning of industrial robot systems more reliable by verifying the programs of robots and logical controllers. This in particular includes optimizing the schedule of the robot systems in order to ensure desired period times as well as conflict-free timetables already in the planning stage. The applicability of the Periodic Event Scheduling Problem proposed by Serafini and Ukovich (SIAM J Discrete Math 2(4):550–581, 1989) is investigated to tackle this cycle time minimization task, and we establish a variant of the classical formulation in order to cover the special characteristics of our scenario. We want to demonstrate how this key element forms a part of a range of developed software tools that support engineers and programmers throughout the commissioning of real-world robot production systems.

Keywords Periodic event scheduling · Mixed-integer programming · Robot systems

✉ Tobias Hofmann
tobias.hofmann@mathematik.tu-chemnitz.de

David Wenzel
david.wenzel.mathematik@t-online.de

¹ Faculty of Mathematics, Chemnitz University of Technology, Chemnitz, Germany

² Leadec Automation and Engineering GmbH, Stuttgart, Germany

³ Institute for Solar-Terrestrial Physics, German Aerospace Center – DLR, Neustrelitz, Germany

1 Motivation

In industrial production, using robots is a versatile mean to ensure manufacturing processes that are reliable and flexible. The technology allows for operating, e.g., with heavy loads, in harmful environments, and at non-ergonomic situations. At the same time, the work is done with high precision and under a reproducible quality. Because of these advantages, robots are widely used in mass production, in particular when several steps are required for obtaining the final product. Ongoing developments also spread the benefits to small series, even down to individual requests.

1.1 A glimpse into application

The typical robot is nothing more than a kind of positioning machine, to which several tools can be attached performing the actual task like handling (pick and place, assembly) or connecting (gluing, welding). Most common are portal-like constructions for getting to every point in (well, a technically limited) space and six-joint kinematics if, in addition to the position, the spatial orientation of the so-called end effector is relevant.

Usually, the robot's movement is described by the trace of a special coordinate system associated with the tool in a chosen world system. In application, this trace is spanned by programmed points, which the robot controller interpolates between in a defined manner—we mean in fact again coordinate systems, to be more precise. Letting the robot do this inverse calculation is the standard, and the alternative way of giving a program for the joints directly is rare due to the loss of imagination. Humans find fitting points commonly offline in CAD simulations and adjust them to reality online directly at the installed machines.

When more than one robot is applied, moving to the same parts can be critical, as their working spaces may overlap. Occupying the same physical space is impossible at the same time, so two robots would collide if they try. Such crashes might impede or even destroy elements of the facility in the worst case or delay production in the best case only. This is expensive and must be prevented by locking the dangerous situations. Basically, one robot simply has to wait for the other.

In general, the robots will not act independently. Though they will follow their respective instructions under own responsibility, the relevant program is assigned by a programmable logic controller (PLC) depending on which actions are necessary for producing the chosen items. The PLC is connected to all the components and, in addition to selecting the robots' working programs, is able to synchronize them on a deeper level via bypassing signals. In this fashion, the robots may interchange binary information through digital input/output channels on every program point. This is typically utilized for safeguarding movable components.

An interlock between two robots' program segments in principle is then established by the following procedure:

When reaching the starting point of its segment, robot 1 asks whether this is free. If not, it waits until it is. But if robot 2 is currently not in its respective crucial segment prone to collision with it, robot 1 will block the segment and

continue its own motion. At the end point of the segment, clearance is given. Since both robots are linked over crossed inputs/outputs, this communication implements the desired behavior.

See Fig. 1 for an illustration. In practice, the core protocol may be extended further for guaranteeing safe functionality even in case of partial component failure, but this won't alter the mathematical point of view.

For installing *robot lines*, some kind of ordinary proceeding has emerged that planners and engineers follow. Well, as one might expect, the geometric positions and process interactions in application tend to be far away from the mathematical understanding of linearity; look at Fig. 4 at the end of the paper to get an impression.

Firstly, based on estimated sales figures, the target time is fixed. This is the interval to be between two outputted final products. Then, according to the amount of required working steps, a decision is made on how many lines or stations are in need for performing different or, when time-consuming, sometimes equal tasks in parallel. With respect to the (by process design separated) stations' manufacturing tasks, a number of robots with fitting tools is prescribed. Consequently, a distribution of the single tasks to the individual robots can be made. Though this seems to include technological information, it is a process more or less driven by economic data or empirical experiences.

Afterwards, more specific decisions are made in detailed planning and simulation. One looks for suitable positions of the robots, given the allocable space and product geometry. Not rarely, a virtual representation is built in a CAD-supported simulation system. This is the entry point for the offline robot programmers. Slightly freer in their actions, they implement motions so that each robot will do its work under the premise of the time fixed at the very beginning. Interactions quite often are respected only rudimentary. The obvious delays like waiting with operation until another robot has inserted the part into the station of course can be considered, but the consequences of all the (also unintended) shared areas in space are hardly to examine in full detail, especially since their extent meanwhile is too immense.

Though simulation systems heavily have been enhanced, both problems have grown too big for being really mastered. Collisions cannot be avoided, since looking for critical segments in a three-dimensional and even animated picture of the robot line is only as good as the human doing it, and hence often erroneous. And predicting the resulting total cycle time is impossible because the interplay is not or not properly implemented. In consequence, lots of work is done on-site during commissioning the machines.

Fig. 1 Scheme for interlocking two robot path segments. The combined movements of two robots can be blocked by getting and setting signals. The effect is that the dashed marked part on the two paths cannot be processed simultaneously, whence collisions are avoided

Online robot programmers adjust the offline programs to the real conditions with construction tolerances in mind and, in combination with the PLC programmers, bring the facility to operation with simultaneously moving robots that are partially interlocked (hopefully correctly). The cycle time is seen at the end, although many efforts are then taken to modify programs for somehow being in time.

1.2 Some backgrounds on available data

While two robots appear to be tractable, the effects in larger production lines no longer can be overseen by men. The complexity of the facilities has grown tremendously over the last years. In the same time, available capacities for adjustments on the real installed robots are more and more limited due to development phases being cut for fast product releases. The previously explained procedure became unsafe, tardy, and costly. But if digital images of planned facilities are already created, one should take benefit of the data instead of doing the same as on the real robot line just in a virtual world. Undoubtedly, the current procedure allows to shift programming jobs to earlier times, but nevertheless the human factor is still determining. With the help of computer algorithms, such systems can be mastered, then even without the susceptibility to missed critical configurations.

Simulating, commissioning, and optimizing robot lines belong to the services Leotec provides for the industry. Special software was developed to attack the problems mentioned above. One of the keys is the determination of collision maps. Recalling the illustration of interlocking a pair of robots given before, this is a two-dimensional representation of all critical constellations. Think of the two robot paths to be straightened and depicted one against the other, i.e., one vertically and the other horizontally. Then, every configuration the robot may take on its path corresponds to one point on the line. And the direct product, i.e., a point in the rectangle spanned by the two lines, stands for the combined constellation of the two robots. The software performs a collision calculation and delivers a visual overview by coloring collision-free constellations green and crash situations differently (cf. Fig. 2).

Roughly speaking, every robot geometry attained on a path of one robot will be calculated against those of the other. Of course, any trivial attempt would not only overstrain humans; also computers are unable to manage the expensive calculations in affordable time. Cleverly designed algorithms are necessary to overcome the hurdles. But we do not go into these details and treat such collision maps as an assumption for further investigations. Clearly, planning interlocks is made easy if one has all the maps of all robot pairs in a line. We want to show how, also based on this information, the minimal total time and a realizing cycle can be determined.

When introducing the term *cycles*, we remind the reader that industrial manufacturing intends to produce the same over and over again. The robots are a major factor, as they can follow their instructions repeatedly. Under the same given parameters, a robot will always react in the same, hence predictable way, which is good for the later presented mathematical models. The programmers link all the components with the central controller and create a relatively stable cycle—up to small perturbations, which are invoked by external influences and are not wanted but also not completely avoidable in real-world mechanical, electrical, or technological processes. In practice, the

Fig. 2 Principle and example of a collision map. The points in the plane are in one-to-one correspondence to combined constellations of two robots in space (left). The color shows whether the two robots may be in the respective position simultaneously (i.e., movements are *green-lit*) or not. The genuine example (right) indicates the practically even more sophisticated range of collision situations, which hints on applicable counter means other than possibly time-reducing interlockings, and a differentiation of robot path points. Those that are reached exactly can be used for waiting, but others serve as an approximation mark on the path and should only be used for signaling. For color references we refer to the online version of the article

usual installation procedure addresses this cycle and is trustworthy only in a small band around it. Larger disturbances may unveil massive security leaks or enforce switching to a different cycle. With the collision maps, these dangers are eliminated, and one is able to plan the cycle in a target-oriented procedure. Ideally, we are talking about a periodic process, which is what mathematics looks onto for search as well as for analysis.

In the following, we will translate the problem into a mathematical formulation that is inspired by previously studied scheduling tasks. A standardized program for optimization is developed and several algorithms for ensuring the claimed structural properties are presented. In the end, the transformation into a variant of the well-known Periodic Event Scheduling Problem (PESP) is suggested for an effective examination.

2 A mathematical model for robot lines

2.1 Reproducing a robot's individual cycle

While asking for a production system's schedule, our objective is to minimize its total cycle time $T > 0$, which should be smaller than the set target in practice. Throughout this paper we are given a finite set of robots R and for each robot $r \in R$ a set of states $V_r := \{1, 2, \dots, n_r\}$ with $n_r \geq 2$ that are collected in

$$V := \bigcup_{r \in R} V_r.$$

These states resemble the programmed points and are to be periodically processed by the robot they belong to. So, the state transitions close a cycle

$$A_r := A(V_r) := \{(1, 2), (2, 3), \dots, (n_r, 1)\}.$$

Furthermore, for each $a = (\alpha, \beta)$ in A_r there are known minimum as well as maximum transition durations $l_a > 0$ and $u_a \geq l_a$ that robot r keeps while passing from state α to state β . Since each state ought to be processed exactly once per period, we can assume $u_a < T$ for each $a \in A_r$ and for each $r \in R$. Actually, the minimum time is fixed by the repeatable robot motion.

The schedule we are looking for should tell us at what time the robots periodically have to start in the states of V while respecting the given bounds on the transitions. More precisely, we seek a schedule $x \in [0, T)^V$ that satisfies all so-called time window constraints

$$l_a \leq (x_\beta - x_\alpha) \bmod T \leq u_a \quad \text{for } a \in A_r, \quad r \in R. \quad (1)$$

Except for the last transition, one has $\beta = \alpha + 1$, recapturing the robot's program flow. By the definition of the modulo operation, the constraint is equivalent to

$$l_a \leq x_\beta - x_\alpha - T \max\{z \in \mathbb{Z} \mid x_\beta - x_\alpha - Tz \geq 0\} \leq u_a,$$

which can be rewritten into

$$l_a \leq x_\beta - x_\alpha + T p_a \leq u_a$$

by defining the periodic offset variable $p_a := -\max\{z \in \mathbb{Z} \mid x_\beta - x_\alpha - Tz \geq 0\}$. Additionally, the condition occurring in this maximum tells us something about the range of p_a , since

$$z \leq \frac{x_\beta - x_\alpha}{T}$$

is required. By using $x \in [0, T)^V$, we conclude that

$$-1 = \frac{0 - T}{T} < \frac{x_\beta - x_\alpha}{T} < \frac{T - 0}{T} = 1,$$

which implies $\max\{z \in \mathbb{Z} \mid x_\beta - x_\alpha - Tz \geq 0\} \in \{-1, 0\}$ or equivalently $p_a \in \{0, 1\}$. Indeed, the time window constraints reflect the desired periodic behaviour of our schedule x . For this, if x satisfies the time window constraint (1), then $x + nT$ satisfies (1) for all $n \in \mathbb{N}$. As already mentioned, each robot $r \in R$ has to run through all its states exactly once per period. Thus, the durations between these states have to add up to the period time T , more precisely, our schedule $x \in [0, T)^V$ has to satisfy

$$\sum_{a=(\alpha, \beta) \in A_r} ((x_\beta - x_\alpha) \bmod T) = T \quad \text{for } r \in R. \quad (2)$$

Proposition 1 *Constraint (2) is equivalent to*

$$\sum_{a \in A_r} p_a = 1 \quad \text{for } r \in R.$$

Proof Constraint (2) requires that

$$\begin{aligned} T &= \sum_{a=(\alpha,\beta) \in A_r} (x_\beta - x_\alpha) \bmod T = \sum_{a=(\alpha,\beta) \in A_r} (x_\beta - x_\alpha + T p_a) \\ &= \sum_{(\alpha,\beta) \in A_r} (x_\beta - x_\alpha) + T \sum_{a \in A_r} p_a = T \sum_{a \in A_r} p_a \quad \text{for } r \in R. \end{aligned}$$

This is satisfied if and only if $\sum_{a \in A_r} p_a = 1$ for $r \in R$. □

A possibility to visualize the structure of our problem is to consider the state set V as the node set of a graph and the state transitions as the associated directed edges. This is illustrated in Fig. 3, where the numbers assigned to the arcs are the known minimum transition durations.

Example 1 The described cycle time minimization problem corresponding to the graph (V, A) from Fig. 3 is given by the mixed-integer program

$$\begin{aligned} &\text{minimize} \quad T \\ &\text{such that} \quad \begin{array}{c} (1,2) \\ (2,3) \\ (3,4) \\ (4,5) \\ (5,1) \end{array} \begin{pmatrix} 2 \\ 1 \\ 3 \\ 2.5 \\ 1 \end{pmatrix} \leq \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ 1 & 0 & 0 & 0 & -1 \end{pmatrix} x + T p < \begin{pmatrix} T \\ \vdots \\ T \end{pmatrix}, \\ &x \in [0, T]^V, \quad \mathbf{1}^\top p = 1, \quad p \in \{0, 1\}^A, \end{aligned}$$

where $\mathbf{1}^\top := (1 \ 1 \ 1 \ 1 \ 1)$. An optimal solution to this problem is $x = (0 \ 2 \ 3 \ 6 \ 8.5)^\top$ with $p = (0 \ 0 \ 0 \ 0 \ 1)^\top$, leading to the optimal value $T = 9.5$. □

Fig. 3 The individual robot cycle. Periodic robot operations (left) can be represented by directed simple closed walks in graph theory (right)

2.2 The conflict matrix

Up to now, we have covered the individual states and transitions of each robot separately. However, our scheduling problem becomes much more interesting if processes of different robots are not executable independently of each other, i.e., when there is the need for one robot letting pass another. As described in the first section, information about the interdependencies between the individual processes is given by collision maps for pairs of robots. Translated into mathematics, for each pair of robots $(r, s) \in R \times R$, we are given a conflict matrix

$$C_{r,s} \in \{0, 1\}^{2n_r \times 2n_s},$$

mapping each of the process constellations of robots r and s to 0 or 1, encoding that this constellation causes no problems in the first case or that it should be prevented by some constraints in the second case. You may have noticed more than two colors in Fig. 2. In the mathematical model, these are combined into only two groups, the good and the bad constellations. The actual assignment is determined by needs of application and can be regarded as given.

Example 2 For two robots r and s , their conflict matrix

$$C_{r,s} = \begin{matrix} r \backslash s & 1 & (1,2) & 2 & (2,3) & 3 & (3,1) \\ \begin{matrix} 1 \\ (1,2) \\ 2 \\ (2,3) \\ 3 \\ (3,4) \\ 4 \\ (4,1) \end{matrix} & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

for instance encodes that robot s must not pass from state 2 to state 3 while robot r is in state 2 (red framed). However, if robot r is in state 1, robot s can pass from state 2 to state 3 without causing any problems (green highlighted entry). $\square$

Notice that the practical information is indeed given for states as well as transitions for taking into account that waiting is only possible in states / program points. Naturally, the conflict information given is symmetric, and robots do not block themselves. More precisely, we have

$$C_{r,s} = C_{s,r}^T \quad \text{and} \quad C_{r,r} = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$$

for each $r \in R$ and each $s \in R$. Accordingly, for setting up the following conflict avoidance constraints, we only run over the pairs $(r, s) \in R \times R$ with $r < s$. Furthermore, these matrices follow some additional properties arising from the scenario.

Definition 1 (*Conflict matrix properties*) Given two robots $r, s \in R$, we refer to the entries of their respective conflict matrix $C_{r,s}$ by c_{ij} for $i \in I := \{1, 2, \dots, 2n_r\}$ and $j \in J := \{1, 2, \dots, 2n_s\}$ and postulate the following properties:

- (i) There is $c_{i,1} = 0$ for each $i \in I$.
- (ii) There is $c_{1,j} = 0$ for each $j \in J$.
- (iii) If $c_{i,k} = 1$ for some $i \in I$ and $k \in \{3, 5, 7, \dots, 2n_s - 1\}$, then also

$$c_{i,k-1} = 1 \quad \text{and} \quad c_{i,k+1} = 1.$$

- (iv) If $c_{k,j} = 1$ for some $k \in \{3, 5, 7, \dots, 2n_r - 1\}$ and $j \in J$, then also

$$c_{k-1,j} = 1 \quad \text{and} \quad c_{k+1,j} = 1.$$

Properties (i) and (ii) mean that each robot always can rest in its first state unaffected by all the other robots. This is also a typical maintenance condition coming from the real programming. Property (iii) says that if some state k must not be attained while robot s is in a specific state or transition, then also transitions in or out of state k ought not to be allowed, i.e., the end points are considered as part of the transition. Finally, (iv) is analogous to (iii). The reason for the last two properties lies in the continuous nature of robot operations and movements. Possible conflicts do not only occur at a point k , but also shortly before attaining k and after leaving it.

2.3 Investigating the matrix structure

Recalling Figs. 1 and 2, the technical realization of one interlocking signal will secure all constellations associated with a 1-block, a submatrix of the collision matrix having only 1-entries. Visually, the critical areas are hidden below a rectangle. As we will see in the next subsection, such a block can be modeled into constraints in a straight forward fashion.

The matrix $C_{r,s}$ from Example 2 fulfills the requirements of Definition 1. However, the distribution of zeros and ones can be much more complicated than in this basic example. To capture all of the possibly strangely shaped areas of 1-entries, likely several rectangles are necessary. In general, one would not aim to lock a collision-free constellation. However, there are constellations that are completely irrelevant, as they never occur in a valid solution. Locking these simplifies the later model, since the shape is altered advantageously. This can be done by preparing the matrix as follows.

Preprocessing procedure

Given a conflict matrix C , submatrices of C of the form

$$B_- = \begin{pmatrix} * & \cdots & * & 1 \\ \vdots & & \vdots & \vdots \\ * & \cdots & * & 1 \\ 1 & \cdots & 1 & 1 \end{pmatrix} \quad \text{or} \quad B_+ = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ 1 & * & \cdots & * \end{pmatrix}$$

can be eliminated where stars stand for arbitrary entries, 0 or 1. They are always substituted by 1-blocks.

In fact, the time window constraints arising from the transitions associated with the columns and rows of C force the related robots to traverse to the right or downwards only. The barrier of 1-entries at the right and the bottom of B_- would lead into a deadlock, once one of the entries of B_- is entered. For the same reason, there is no possibility to reach one of the constellations represented by the entries in B_+ unless moving backwards, which is prohibited. Thus, we can recursively replace submatrices like these by 1-blocks of fitting size.

The preprocessed conflict matrix keeps the properties of Definition 1, and we are interested in decomposing it into blocks for passing to inequality constraints. As signal information shall only be exchanged at program points, there is an additional requirement we want to impose to the obtained 1-blocks. Remember that points are resembled by odd and segments by even indices.

Proposition 2 *The 1-entries of a conflict matrix C can be covered by 1-blocks*

$$\begin{pmatrix} c_{11} & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & c_{mn} \end{pmatrix},$$

where the 1-entries c_{11} as well as c_{mn} are in an even column and row of C .

Proof Without any doubt, the 1-entries can be distributed over a tight collection of 1-blocks. The question only is why the indices of the corners can be chosen even. So, take an arbitrary block of the cover and look at the borders of the rectangle of 1-entries inside the conflict matrix. Due to properties (i) and (ii), these never are in the first row or column. Should they be in the last row or column, the index already is even by definition. In any other case, suppose the border would be located at an odd number. Because of properties (iii) and (iv), both neighboring row or column segments bear 1-entries. Consequently, the rectangle may be enlarged by decreasing the left/top or increasing the right/bottom border's index by one. In doing so, overlaps with other rectangles may be created, but the cover needn't be disjoint. $\square$

Summarizing, decomposing the collision information leads us to the following subproblem.

Definition 2 (*Set cover problem*) Given a finite set N and $\mathcal{I} \subset \mathcal{P}(N)$, a cover is a subset $\mathcal{I} \subset \mathcal{I}$ whose elements' union is N . The set cover optimization problem is to find a cover with minimum cardinality.

While in general the decision version of the set cover problem is one of the classical 21 NP -complete problems (see Karp 1972) and its optimization version is known to be NP -hard (see Korte and Vygen 2012), the situation facing us has a more promising structure. In our case, N is the set of all 1-entries of a conflict matrix, and we are to cover them with a preferably few 1-blocks. Of course, one always could cover every entry with a small block, but minimizing the number of 1-blocks also minimizes the number of integral variables which are needed later for setting up the arising constraints. In the next step, a suitable cover as in Proposition 2 can be computed.

Cover algorithm

Input: preprocessed conflict matrix $C = (c_{ij})_{i,j=1}^{2n_r, 2n_s}$

Output: set of 1-blocks that cover 1-entries of C

```

1: for  $i = 2, 4, \dots, 2n_r$  do
2:   for  $j = 2, 4, \dots, 2n_s$  do
3:     if  $c_{ij} = 1$  then
4:       calculate (and store) the 1-block

```

$$\begin{pmatrix} c_{ij} & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & c_{mn} \end{pmatrix} \quad \text{with maximal } m \text{ and } n.$$

```

5:       set  $c_{kl} = 0$  for  $k = i, \dots, m$  and  $l = j, \dots, n$ 
6:       restore (iii) and (iv), and apply the preprocessing procedure to  $(c_{ij})_{k=i, l=j}^{m, n}$ 
7:     end if
8:   end for
9: end for

```

Step 5 of the algorithm overwrites areas of C that have been covered in step 4 and is supposed to cause the algorithm to terminate. However, this step might temporarily destroy the conflict matrix properties (iii) or (iv). This is resolved within step 6.

Proposition 3 The entry c_{ij} from step 3 and 4, which is set to 0 within step 5, cannot be set to 1 again in step 6.

Proof At first, we convince ourselves that restoring (iii) and (iv) does no harm. Since m and n are chosen maximal within step 4, we have $c_{i,n+1} = 0$ and $c_{m+1,j} = 0$. Because the input in step 4 fulfills properties (iii) and (iv), the entry c_{mn} lies in an even row and column of C . Thus, row $m + 1$ and column $n + 1$ are uneven and consequently $c_{m+1,j+1} = 0$ and $c_{i+1,n+1} = 0$. Otherwise, properties (iii) or (iv), respectively, would imply $c_{i,n+1}$ or $c_{m+1,j}$ to be nonzero—which is not the case, as we have seen. Since (iii) and (iv) only affect neighboring columns or rows, their application in step 6 does not change any value in column i or row j and therefore cannot set c_{ij} to 1.

On the other hand, if c_{ij} is set to 1 because of the preprocessing procedure, this would require a constellation

$$\begin{pmatrix} c_{ij} & \cdots & 1 & \star & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & \cdots & c_{mn} & \vdots & \vdots \\ \star & \cdots & \cdots & \star & \vdots \\ 1 & \cdots & \cdots & \cdots & 1 \end{pmatrix},$$

which is only possible if the input in step 4 was not preprocessed or m and n were not chosen maximal within step 4. $\square$

Example 3 For illustrating the cover algorithm, regard a typically shaped connected area of 1-entries in a preprocessed conflict matrix:

Beginning at the top left corner, an as large as possible rectangle is selected and the corresponding entries are annihilated. The remaining area is “completed” by the preprocessing procedure, resulting in a new unique top left corner. We obtain another rectangle, and the algorithm is iterated once more.

Clearly, one must use at least as many blocks in a cover than the area has “stairs” either horizontally or vertically; in the example there are three to the right or two downwards. As the algorithm maximally fills these, the minimum number of blocks is obtained. $\square$

2.4 Modeling the scheduling problem

As the cover algorithm works correctly, given two robots $r, s \in R$ with $r < s$ as well as their conflict matrix $C_{r,s}$, we only focus on 1-blocks

$$\begin{array}{cc}
 (\gamma, \gamma') & (\delta, \delta') \\
 (\alpha, \alpha') & \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \\
 (\beta, \beta') &
 \end{array}$$

while setting up the corresponding collision avoidance constraints. This 1-block requires that robot r only starts in state α if robot s has reached state δ' , and robot s only starts in state γ if robot r has reached state β' .

Recall Fig. 1 for convincing you that the technical interlock in reality is implemented this way. Formally, it is expressed by

$$l_{(\delta, \delta')} \leq (x_\alpha - x_\delta) \bmod T \quad \text{and} \quad l_{(\beta, \beta')} \leq (x_\gamma - x_\beta) \bmod T.$$

Following the same reasoning as at this section's beginning, this can be rewritten into

$$l_{(\delta, \delta')} \leq x_\alpha - x_\delta + T p_{(\delta, \alpha)} \quad \text{and} \quad l_{(\beta, \beta')} \leq x_\gamma - x_\beta + T p_{(\beta, \gamma)}$$

with new integral variables $p_{(\delta, \alpha)}$ and $p_{(\beta, \gamma)}$. Again one could also introduce upper bounds less than T .

Furthermore, the operations excluding each other—i.e., the paths vertically and horizontally through the 1-block—represent movements of two robots that happen at different times in the overall cycle of the facility. Hence, they have to be executable within the period time T . In other words, the total duration of all transitions in the cycle-like *blocking circuit*

$$D = D(r, s, \alpha, \beta, \gamma, \delta) := \{(\gamma, \gamma'), \dots, (\delta - 1, \delta), \underline{(\delta, \alpha)}, (\alpha, \alpha'), \dots, (\beta - 1, \beta), \underline{(\beta, \gamma)}\}$$

has to be T . Analogous to Proposition 1 on the robot cycle constraint (2), this can be guaranteed by the constraint

$$\sum_{a \in D} p_a = 1 \quad \text{for each 1-block in } C_{r,s}, \quad r, s \in R, \quad r < s. \quad (3)$$

Remember that α' , the direct successor of α in the robot program, more or less is $\alpha + 1$ modulo the number of states. The added transitions (underlined above) break out of this scheme. The introduced inequalities with the variables $p_{(\delta, \alpha)}$ and $p_{(\beta, \gamma)}$ create a pair of new edges between two robot cycles weighted similarly to the robot state transitions. In this sense, Fig. 1 anticipates the mathematical model construction when drawing arrows from output to input, which carry the query–release interchange for safeguarding two robot path segments. The blocking circuit D then moves along the critical segment on the first robot's cycle, switches to the second's cycle for its segment (dashed arrow in Fig. 1), and again after some time returns to the first one via the new connection.

An example of how the constructed graph looks for real instances will be given in the end in Fig. 4. Finally, there may be further time window constraints arising from certain technical precedences. However, those are structurally the same as the ones already discussed. In summary, all our constraints fit into a framework which was first suggested by Serafini and Ukovich (1989).

Definition 3 (*Periodic Event Scheduling Problem / PESP*) Given a directed graph $G = (V, A)$ as well as vectors $l, u \in \mathbb{N}^A$, the task is to find vectors $x \in [0, T)^V$ and $p \in \{0, 1\}^A$ such that

$$l \leq B^T x + T p \leq u,$$

where B is the node-arc incidence matrix of G .

The only extensions that our scenario requires are the constraints (2) and (3) as well as the objective to minimize T . For all practical needs, the time may be considered discrete. Actually, robot controllers calculate the joint movements with respect to an interpolation interval, which is a kind of smallest time step. Consequently, by some scaling, one can even assume $T \in \mathbb{N}$. This is an unmentioned, but generally desired assumption in Definition 3.

It is known that the PESP is NP-complete; see Odijk (1994) and Nachtigall (1996) for two different proof ideas. The difficulty results from the periodic offsets; Liebchen (2006, Chapter 6) gives a more detailed analysis. However, the number of integral variables can be reduced to $v = |A| - |V| + 1$ by using the equivalent cycle periodicity formulation (see Nachtigall 1994), which is obtained by introducing the variables $y_a := x_\beta - x_\alpha + T p_a$ for each $a = (\alpha, \beta) \in A$, so-called periodic tensions. Eventually, restricting to a basis, this leads to the adapted standard formulation on which our implementations are based on.

Definition 4 (*Cycle periodicity formulation*) Given a directed graph $G = (V, A)$ as well as vectors $l, u \in \mathbb{N}^A$, the task is to find vectors $y \in [0, T)^A$ and $p \in \mathbb{Z}^v$ such that

$$\begin{aligned} \Gamma^T y &= T p, \\ l &\leq y \leq u, \end{aligned}$$

where Γ is an arc-cycle incidence matrix containing an integral cycle basis of G .

For details on cycle bases we refer to the survey by Kavitha et al. (2009). In our application, Γ also has to comprise the cycles A_r for $r \in R$ as well as the circuits arising from all 1-blocks in each $C_{r,s}$ for $r, s \in R$, and we are to include the constraints (2) and (3).

The PESP and the alternative problem have been studied for different applications with periodic behavior. A classical one is the determination of timetables for railway networks; see Liebchen and Möhring (2007) for model aspects and Liebchen and Peeters (2009) on how integral cycle bases are related to this setting. The parallels of trains and robots are strong. Both follow an individual

cycle—driving from station to station vs. moving from state to state. But, they are synchronized somehow through shared resources that must be occupied at different times—the same track in the network vs. the same physical three-dimensional space. Our interlock necessity of two path segments corresponds to single-track parts that two trains shall use on their route.

The graphs generated by the two applied settings naturally look a bit different. While railway networks have lots of stations (also counting crossings) with limited available tracks, robot lines possess comparably few states that have much more possibilities for connections. The deciding difference lies in our intentions in regard of the total time T . Trains have a fixed period (usually hourly for large distances and about 10 minutes for local lines), and one is interested into getting an admissible solution. In contrast, production systems ought to run as fast as possible, so we are confronted with an optimization problem. Note that our additional objective of minimizing T causes nonlinearities in the window constraints. However, these can be resolved by treating the related feasibility problem and utilizing binary search for the minimum period time T . Indeed, in our scenario this approach works out, since occurring upper bounds for $a \in A$ are of the form $u_a = T - \sum_{a \in K} l_a$ for some index set $K \subset A$. So, if the above formulation is solvable for some T , then it is also solvable for any $T' > T$, which enables us to use binary search. Finally, the conflict matrix properties (i) and (ii) ensure that it is possible to let the robots traverse through their states one after the other, so a global upper bound and therefore a reasonable starting point for the search is given trivially by

$$T = \sum_{r \in R} \sum_{a \in A_r} l_a.$$

3 Conclusions

3.1 Feedback from real-world data

The more detailed treatment of model properties and solution behavior will be subject of another article (Helmberg et al.). The performance of the contoured *IPESP*, though remaining *NP*-complete, is shown to behave reasonably well. For a first illustration of the applicability in practice, we want to take a look onto a robot line as shown in Fig. 4. This is a prototypic station, here especially for assembling side parts of cars. The example produces a graph with 448 vertices and 821 edges. About 50 program points per robot can be expected, but this may vary a lot (compare the visible cycles around R1 and R2 in the right picture). Little more than half of the edges are part of the robot cycles, while 354 edges are originated in reconstructing the interdependencies. That is, we have 177 collision-avoiding 1-blocks. The distribution usually is not homogeneous.

The graph clearly reflects the layout in forming two highly connected components. These two are linked with only few edges, since the underlying operation from R5 to R6 is just a handing over of the production item. There are also 19 edges

Fig. 4 Exemplarily layout of a robot line and the corresponding graph. In the line contoured at the left, large items are transported by robots (here shown as dark circles) from the entry (usually another similar line with other processes) to deposit or locator stations (indicated by light rectangles), at which robots will do their work (like welding or glueing) depending on the attached tool. A robot with a gripper may even perform actions at a non-movable tool (light triangles in this sketch). After several such manufacturing steps, the finished item is stored in a container for transport. Stations where robots are working on are dominated by lots of interference conditions. One can imagine how close they are positioned next to each other. The resulting high complexity poses challenges on programmers

coming from technology. Well, obviously R2 may only start operation after R1 put in the item, and R5 must wait until R2 is finished. Unlike the block constraints, the precedence constraints need not to appear in pairs.

Finding the optimal cycle took approximately 17.2 seconds, which is very good in view of the practical requirements in the phases from planning to commissioning. With this information, one can enforce the calculated cycle in the real facility or look for potentials of further (let's call it) *optimization*, e.g., if mandated by failing the targeted time.

The cycles obtained via our model are in a graph-theoretic representation. The users that shall take advantage later on are the programmers from simulation. They are not confronted with the internal structures, but they are given an evaluation in an appropriate form. Well-known pictures are bar diagrams for the robots' operation time spans (Fig. 5 is one belonging to the example). These are usually estimated in

Fig. 5 Bar diagrams as visual timetable representation. A full cycle is shown in only one snapshot. The bars tell when a robot is active, so (expected or unexpected) stops can easily be identified by white gaps. Different steps in a robot's schedule can be colored differently and drawn side-by-side (left), or in several lines (more detailed extract on the right). The latter is widely used, and such Gantt charts are often generated arduously in the beginning with common table manipulation office suites. But the initial plan and the real facility often have only little in common

long-winding manual efforts, as noted, long before any particular robot motion is thought of. Now, the actual to-be-expected diagrams can be obtained directly from the calculated cycle's start and arrival times x_α and $x_\alpha + l_{(\alpha,\alpha')}$. The cycle time here is less than a minute, and the computed solution moreover is about a second faster than the naturally establishing cycle. In this case, a better period can be achieved if a specific robot is stopped shortly for letting another pass. Such a controlled delay would not happen on its own.

3.2 Some thoughts on extensions

The view on the facility as a whole is a promising tool for identifying bottlenecks in order to make sure the desired time can be achieved. An on-site programmer trying to find points to attack can only examine concrete situations by his own eyes, and more hidden influences equally responsible for the current cycle time are easily missed. Recording the full real cycle normally is hard work, and being able to get it in advance enables for an in-depth analysis. The mathematical form of the timetable moreover will automatize parts of this process, as for instance indicating active constraints will unveil bottlenecks. So, we created a pivotal part of the planning software tools, which can be ported to the realization and optimization phases.

When establishing the model, we turned the given multi-color collision map into a 0-1-matrix. This was done by neglecting the presence of potentially collision-free areas—likely with only little adjustments needed to achieve this. Currently, one can determine the cycle of another instance, where more free space is granted through not generally blocking yellow areas. Just assign them 0 instead of 1. A plain comparison similar to a bottleneck analysis then demonstrates the possibilities and shows the crucial points for additional cycle time reductions. For broader integration, the model and respective algorithms can even be studied for extension to directly incorporate the distinction of absolutely necessary and preferable but non-binding constraints.

Also other relevant aspects emerging from application deserve taking care of. Some facilities for example include human workers in addition to the robots. From the mathematical perspective, these may be treated just like robots. A deciding difference for analysis could be the unsure working time. Their influences already can be studied by calculating with various values, but also an adaption to the model with a range or probability distribution replacing the fixed transition times might be considered in the future.

Acknowledgements We are greatly indebted to Leadec Industrial Services for providing abstract data oriented on real-world production systems, preparing graphics, and substantiating developed approaches and computed results.

Funding This research is supported by the European Union through ERDF Grants SAB 100206298 and 100206299. Open Access funding is provided by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article

are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Helmberg C, Hofmann T, Wenzel D (2020) Periodic event scheduling for automated production systems. Preprint available online via https://www.tu-chemnitz.de/mathematik/discrete/papers/Preprint_PESP_2020.pdf
- Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of computer computations, pp 85–103
- Kavitha T, Liebchen C, Mehlhorn K, Michail D, Rizzi R, Ueckerdt T, Zweig KA (2009) Cycle bases in graphs characterization, algorithms, complexity, and applications. *Comput Sci Rev* 3(4):199–243
- Korte B, Vygen J (2012) Combinatorial optimization. Springer, Berlin
- Liebchen C (2006) Periodic timetable optimization in public transport. *dissertation.de*
- Liebchen C, Möhring R (2007) The modeling power of the periodic event scheduling problem: railway timetables and beyond. In: Algorithmic methods for railway optimization, pp 3–40
- Liebchen C, Peeters L (2009) Integral cycle bases for cyclic timetabling. *Discrete Optim* 6(1):98–109
- Nachtigall K (1994) A branch-and-cut approach for periodic network programming. In: Hildesheimer Informatik Berichte 29
- Nachtigall K (1996) Cutting planes for a polyhedron associated with a periodic network. Technical Report IB 112-96/17, Deutsches Zentrum für Luft- und Raumfahrt
- Odijk MA (1994) Construction of periodic timetables. Pt. 1. A cutting plane algorithm. Report 94-61, TU Delft
- Serafini P, Ukovich W (1989) A mathematical model for periodic scheduling problems. *SIAM J Discrete Math* 2(4):550–581

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.