

Studies in Systems, Decision and Control 241

Wojciech Bożejko  
Grzegorz Bocewicz *Editors*

# Modelling and Performance Analysis of Cyclic Systems

 Springer

# **Studies in Systems, Decision and Control**

Volume 241

## **Series Editor**

Janusz Kacprzyk, Systems Research Institute, Polish Academy of Sciences,  
Warsaw, Poland

The series “Studies in Systems, Decision and Control” (SSDC) covers both new developments and advances, as well as the state of the art, in the various areas of broadly perceived systems, decision making and control—quickly, up to date and with a high quality. The intent is to cover the theory, applications, and perspectives on the state of the art and future developments relevant to systems, decision making, control, complex processes and related areas, as embedded in the fields of engineering, computer science, physics, economics, social and life sciences, as well as the paradigms and methodologies behind them. The series contains monographs, textbooks, lecture notes and edited volumes in systems, decision making and control spanning the areas of Cyber-Physical Systems, Autonomous Systems, Sensor Networks, Control Systems, Energy Systems, Automotive Systems, Biological Systems, Vehicular Networking and Connected Vehicles, Aerospace Systems, Automation, Manufacturing, Smart Grids, Nonlinear Systems, Power Systems, Robotics, Social Systems, Economic Systems and other. Of particular value to both the contributors and the readership are the short publication timeframe and the world-wide distribution and exposure which enable both a wide and rapid dissemination of research output.

\*\* Indexing: The books of this series are submitted to ISI, SCOPUS, DBLP, Ulrichs, MathSciNet, Current Mathematical Publications, Mathematical Reviews, Zentralblatt Math: MetaPress and Springerlink.


More information about this series at <http://www.springer.com/series/13304>


Wojciech Bożejko · Grzegorz Bocewicz  
Editors

# Modelling and Performance Analysis of Cyclic Systems

 Springer

*Editors*

Wojciech Bożejko   
Department of Control Systems  
and Mechatronics, Faculty of Electronics  
Wrocław University of Science  
and Technology  
Wrocław, Poland

Grzegorz Bocewicz   
Department of Computer Science  
and Management, Faculty of Electronics  
and Computer Science  
Koszalin University of Technology  
Koszalin, Poland

ISSN 2198-4182                      ISSN 2198-4190 (electronic)  
Studies in Systems, Decision and Control  
ISBN 978-3-030-27651-5              ISBN 978-3-030-27652-2 (eBook)  
<https://doi.org/10.1007/978-3-030-27652-2>

© Springer Nature Switzerland AG 2020

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG  
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Preface

Production systems implementing multi-assortment production, in large quantities, with slowly changing in time, product mix, manufacture products exclusively in a cyclic manner. Similar issues also appear in computer systems, for instance—embedded ones. Modern systems of these types satisfy market demand either by a uniform production (or processing) in time, whose range varies irregularly according to market demand, or by cyclic output providing the right mix of assortments/processes. The latter approach is more economically attractive as it eliminates or reduces the size of the stored finished goods. In addition, it provides a systematic inventory replenishment of relatively small amount of goods at suppliers and generates systematic demand for intermediates, raw materials from deliverers. The above system also simplifies the process of supply chain management. It also enables relatively easy detection of defects that may indicate a deterioration of the quality of the production system parameters and/or manufactured products. Optimization of the operation of such a system is reduced to minimizing of the cycle period, for a fixed mixture of tasks (products, processes) in the cycle, which results in increased system capacity and improvement of machine utilization. Thus, recently, one can observe a significant increase of interest in the problems of cyclic tasks scheduling theory. For they are usually important and difficult, mostly NP-hard problems, from the standpoint of not only theory but also practice.

Cyclic problems are unique and little researched. They belong to a subclass of scheduling problems, as in fact they relate to the so-called irregular criterion. Cyclic problems are object of interest primarily due to their strong practical importance and difficulty in obtaining adequately efficient algorithms solving particular cases with additional constraints arising from manufacturing practice.

Above-mentioned topics should be of great interest to researchers in computer science, operations management, production control, as well as practicing managers and engineers. Featuring a balance between state-of-the-art research and practical applications, this monograph provides a forum for contributions that cover the main research challenges related to the cyclic modelling, development and validation of

concurrently acting distributed production systems and processes. After a strict review, following scientific findings from researchers around the Europe have been accepted.

The book is divided into four parts. First four chapters consider problems modelling by a graph approach. Chapters “[Blockage-Free Route Planning for In-Plant Milk-Run Material Delivery Systems](#)” and “[Coordination of Cyclic Motion Processes in Free-Ranging Multiple Mobile Robot Systems](#)” deal with cyclic route planning issues. Chapters “[Conflict Avoidance Within Max-Plus Fault-Tolerant Control: Application to a Seat Assembly System](#)” and “[Max-Plus Algebraic Modelling of Cyclical Multi-assortment Manufacturing System](#)” propose the usage of max-plus algebra for cyclic systems modelling. Chapter “[Incorporating Automatic Model Checking into GPenSIM](#)” presents Petri nets application for cyclic systems representation.

Chapter “[Cyclic Data Flows in Computers and Embedded Systems](#)” of Hanen C. and Munier-Kordon A. deals with the issue of cyclic DataFlow in computer and embedded systems. The chosen modelling method is Synchronous DataFlow Graphs as a simple model of computation introduced for the description of Digital Signal Processing Applications. The chapter aims at presenting theoretical results as well as practical applications in context of cyclic scheduling problems.

Chapter “[Cyclic Two Machine Flow Shop with Disjoint Sequence-Dependent Setups](#)” of Bożejko W., Smutnicki C., Uchroński M. and Wodecki M. considers the problem of cyclic scheduling on two machines with resource constraints for setups, concerning a single team that can perform setup between operations on a machine. Such a limitation significantly impedes the considered issue because the solution is represented here not only by the order of performing jobs, but also by the route of the setup team, i.e. the order in which the team makes setups of machines. Linear programming model is proposed, coded in AMPL modelling language and tested in Gurobi solver.

Chapter “[Cyclic Scheduling in the Manufacturing Cell](#)” of Bożejko W., Pempera J., Smutnicki C. and Wodecki W. is focused on jobs scheduling performed by machines and by an operator in automated manufacturing cells that have a large volume of cyclic production. The purpose of scheduling is to set a cyclical schedule that minimizes production cycle time. The chapter presents a genuine model of the considered problem enabling effective determination of cycle time for any sequence of operations in the cell. There is also an algorithm proposed that determines the sequence and schedule of jobs minimizing the production cycle time.

Chapter “[On Estimating LON-Based Measures in Cyclic Assignment Problem in Non-permutational Flow Shop Scheduling Problem](#)” of Gnatowski A. and Niżyński T. deals with Fitness Landscape Analysis, which has provided a variety of new approaches to analyse problem instances, focusing on Data-Driven approach to problem-solving. The method proposed is based on Local Optima Networks methodology—a compact representation of a search space from the perspective of optimization algorithms. The impact of the number of samples taken, on the obtained LON metrics for Cyclic Assignment Problem in non-permutational Flow Shop Scheduling Problem, is analysed. The results suggest a strong relation between the measure values and sampling effort.

In chapter “[Coordination of Cyclic Motion Processes in Free-Ranging Multiple Mobile Robot Systems](#)”, Roszkowska E. proposed the control architecture that implements deadlock/collision avoidance conditions for Multiple Mobile Robot System (MMRS). Considered problem concerns the cyclic routing of asynchronous mobile robots sharing a common 2D motion space. A model of the feasible dynamic behaviour of the robot system is then obtained by mapping the distinguished Resource Allocation System (RAS) into a Deterministic Finite State Automaton (DFSA). Based on this model, a control architecture that implements the described control logic and combines it with the priority control, thus receiving a flexible controller for MMRS, is finally proposed.

In chapter “[Blockage-Free Route Planning for In-Plant Milk-Run Material Delivery Systems](#)”, Bocewicz G., Nielsen I. and Banaszak Z. discussed how to efficiently support routing and scheduling decisions regarding the movement of vehicles in an in-plant milk-run delivery system. The problem under study, called the Multi-Trip and Multi-Cycle Pickup and Delivery Problem with Time Windows and Congestion-Free Traffic, can be viewed as extension of the pickup and delivery problem with time windows in which multiple tigger trains travel along closed-loop congestion-free routes in different cycles. To solve this kind of problem, the recursive constraint satisfaction problem is formulated. Its solution provides solutions that can minimize both vehicle downtime and the takt time of the production flow.

In chapter “[Conflict Avoidance Within Max-Plus Fault-Tolerant Control: Application to a Seat Assembly System](#)”, Witczak M., Majdzik P., Lipiec B. and Stetter R. focused on the flexibility of manufacturing and assembly systems allowing for more efficient activities aiming at following the dynamically evolving markets. In that context, proposed max-plus algebra model allows to predict delivery times of products which are customized to products’ requirements in the cyclic manufacturing system and finally to evaluate the cost related to the reconfiguration of the system.

Chapter “[Max-Plus Algebraic Modelling of Cyclical Multi-assortment Manufacturing System](#)” of Stańczyk J. is focused on an analysis of the behaviour of multi-assortment production systems represented in terms of as Discrete Event Systems (DES) and modelled through the system state equations in the max-plus algebra formalism. A number of phenomena that have a direct impact on the behaviour of systems, such as ending the production of one product or launching, in an already existing production system, the production of an additional and a new product are analysed.

In chapter “[Incorporating Automatic Model Checking into GPenSIM](#)”, Davidrajuh R., Skolud B. and Krenczyk D. presented the General-purpose Petri Net Simulator (GPenSIM) implementing the Activity-Oriented Petri Nets (AOPN) representation. GPenSIM being a tool for modelling, simulation, performance evaluation and control of discrete event systems are used to avoid unexpected failures in large-scale manufacturing systems. Its model checking functions are illustrated through examples following cyclic production systems.



In the editors' intention, this book has a monographic character focusing on cyclic systems, wherein each chapter has an independent character and it is written by authors who have a well-established position in the field of issues related to cyclic systems. It collected in a systematized way significantly broadened results on the subject of methods of modelling and solving difficult issues of optimization and manufacturing which appears in IT and control systems. The aim of this book is to familiarize the reader with the contemporary methodology of cyclic systems modelling in application to the issues of industrial engineering.

Wrocław, Poland  
Koszalin, Poland  
April 2019

Wojciech Bożejko  
Grzegorz Bocewicz

**Acknowledgements** Some of the studies presented in the book were partly funded by the National Science Centre of Poland, grant OPUS No. DEC 2017/25/B/ST7/02181—this refers to chapters “Cyclic Two Machine Flow Shop with Disjoint Sequence-Dependent Setups”, “Cyclic Scheduling in the Manufacturing Cell” and “On Estimating LON-Based Measures in Cyclic Assignment Problem in Non-permutational Flow Shop Scheduling Problem”. Chapters “Cyclic Data Flows in Computers and Embedded Systems” and “Coordination of Cyclic Motion Processes in Free-Ranging Multiple Mobile Robot Systems”—“Incorporating Automatic Model Checking into GPenSIM” were written for the exclusive needs of this book. Calculations from chapter “Cyclic Two Machine Flow Shop with Disjoint Sequence-Dependent Setups” have been carried out using resources provided by Wrocław Centre for Networking and Supercomputing, grant No. 096.

# Contents

## Graph Modeling of Cyclic Systems

<b>Cyclic Data Flows in Computers and Embedded Systems</b> . . . . .	3
Claire Hanen and Alix Munier-Kordon	

<b>Cyclic Two Machine Flow Shop with Disjoint Sequence-Dependent Setups</b> . . . . .	31
Wojciech Bożejko, Czesław Smutnicki, Mariusz Uchroński and Mieczysław Wodecki	

<b>Cyclic Scheduling in the Manufacturing Cell</b> . . . . .	49
Wojciech Bożejko, Jarosław Pempera, Czesław Smutnicki and Mieczysław Wodecki	

<b>On Estimating LON-Based Measures in Cyclic Assignment Problem in Non-permutational Flow Shop Scheduling Problem</b> . . . . .	63
Andrzej Gnatowski and Teodor Niżyński	

## Cyclic Route Planning

<b>Coordination of Cyclic Motion Processes in Free-Ranging Multiple Mobile Robot Systems</b> . . . . .	87
Elzbieta Roszkowska	

<b>Blockage-Free Route Planning for In-Plant Milk-Run Material Delivery Systems</b> . . . . .	105
Grzegorz Bocewicz, Izabela Nielsen and Zbigniew Banaszak	

## Max-Plus Algebra for Cyclic Systems Modeling

<b>Conflict Avoidance Within Max-Plus Fault-Tolerant Control: Application to a Seat Assembly System</b> . . . . .	135
Marcin Witczak, Paweł Majdzik, Bogdan Lipiec and Ralf Stetter	

**Max-Plus Algebraic Modelling of Cyclical Multi-assortment Manufacturing System** ..... 159  
Jarosław Stańczyk

**Petri Nets for Cyclic Systems Modeling**

**Incorporating Automatic Model Checking into GPenSIM** ..... 175  
Reggie Davidrajuh, Bozena Skolud and Damian Krcencyk

**Index** ..... 189

# Editors and Contributors

## About the Editors



**Wojciech Bożejko** is an Associate Professor at Wrocław University of Science and Technology. He received his M.Sc. from the University of Wrocław Institute of Computer Science in 1999, his Ph.D. from Wrocław University of Technology Institute of Engineering Cybernetics in 2003 and his D.Sc. (habilitation) from Wrocław University of Technology Faculty of Electronics in 2011. Since 2013, he has been an Associate Professor at the Wrocław University of Science and Technology (Politechnika Wrocławska), Faculty of Electronics, Department of Control Systems and Mechatronics. He is the author of over 230 peer-reviewed papers published in journals and conference proceedings, on parallel processing, scheduling and optimization. His research interests include methods for solving NP-hard problems, especially parallel algorithms, GPU computing, scheduling and discrete optimization, but also group theory and free probability.



**Grzegorz Bocewicz** is an Associate Professor at the Faculty of Electronics and Computer Science, Koszalin University of Technology, Poland. He received his M.Sc. degree in Telecommunications from Koszalin University of Technology, Poland, and his Ph.D. and D.Sc. degrees in Computer Sciences from Wrocław University of Technology, Poland, in 2006, 2007 and 2014, respectively. Since 2016, he has been Dean of the Faculty of Electronics and Computer Science at Koszalin University of Technology. His research interests are in the modelling and design of decision support systems, methods for advanced planning and scheduling, constraints programming techniques, modelling and analysing systems of concurrent cyclic processes, operational research techniques, artificial methods, theory of dynamic discrete event systems, and project portfolio prototyping under uncertain constraints. He is the author or co-author of over 60 journal articles, 3 books, 60 book chapters and 36 peer-reviewed conference papers.

## Contributors

**Zbigniew Banaszak** Faculty of Electronics and Computer Science, Koszalin University of Technology, Koszalin, Poland

**Grzegorz Bocewicz** Faculty of Electronics and Computer Science, Koszalin University of Technology, Koszalin, Poland

**Wojciech Bożejko** Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

**Reggie Davidrajuh** Department of Electrical & Computer Engineering, University of Stavanger, Stavanger, Norway

**Andrzej Gnatowski** Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

**Claire Hanen** UPL, Université Paris-Nanterre, Nanterre, France;  
Sorbonne Université, CNRS, LIP6, Paris, France

**Damian Krenczyk** Faculty of Mechanical Engineering, Institute of Engineering Processes Automation and Integrated Manufacturing Systems, Silesian University of Technology, Gliwice, Poland

**Bogdan Lipiec** Institute of Control and Computation Engineering, University of Zielona Góra, Zielona Góra, Poland

**Paweł Majdzik** Institute of Control and Computation Engineering, University of Zielona Góra, Zielona Góra, Poland

**Alix Munier-Kordon** Sorbonne Université, CNRS, LIP6, Paris, France

**Izabela Nielsen** Department of Materials and Production, Aalborg University, Aalborg Øst, Denmark

**Teodor Niżyński** Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

**Jarosław Pempera** Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

**Elzbieta Roszkowska** Department of Cybernetics and Robotics, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

**Bożena Skolud** Faculty of Mechanical Engineering, Institute of Engineering Processes Automation and Integrated Manufacturing Systems, Silesian University of Technology, Gliwice, Poland

**Czesław Smutnicki** Department of Computer Engineering, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

**Jarosław Stańczyk** Department of Genetics and Animal Breeding, Wrocław University of Environmental and Life Sciences, Wrocław, Poland

**Ralf Stetter** Faculty Mechanical Engineering, University of Applied Sciences Ravensburg-Weingarten, Weingarten, Germany

**Mariusz Uchroński** Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

**Marcin Witczak** Institute of Control and Computation Engineering, University of Zielona Góra, Zielona Góra, Poland

**Mieczysław Wodecki** Department of Telecommunications and Teleinformatics, Faculty of Electronics, Wrocław University of Science and Technology, Wrocław, Poland

# **Graph Modeling of Cyclic Systems**

# Cyclic Data Flows in Computers and Embedded Systems



Claire Hanen  and Alix Munier-Kordon

**Abstract** Synchronous DataFlow Graphs (SDF in short) is a simple model of computation introduced for the description of Digital Signal Processing Applications. This formalism is today widely used to model embedded parallel applications. This chapter aims at presenting a panorama of theoretical results and practical applications in connection with cyclic scheduling problems. We first recall that the execution of a SDF can be seen as a set of cyclic dependant tasks. The structure of precedence constraints, important dominance properties and simplifications of the SDF are then presented. For the special case of uniform precedence graph, periodic schedule are dominant and the maximum throughput can be polynomially evaluated. Main results on the resource constrained problem are presented, followed by a more recent problem issued from sensor networks. In the general case, the existence of a polynomial-time algorithm to evaluate the maximum throughput of a SDF is a challenging question. However, the determination of a periodic schedule of minimum period is a polynomial problem, and many authors limit their study to this class of schedule to express optimization problems as the total buffer minimization or to evaluate the latency of a real-time periodic system.

## 1 Introduction

This chapter addresses cyclic scheduling problems issued from the control of data flows in computers, embedded systems or sensor networks. Although in various context, parts and data may induce the same theoretical scheduling problems, we focus here on specific models and constraints. We point out analogies with produc-

---

C. Hanen (✉)  
UPL, Université Paris-Nanterre, Nanterre, France  
e-mail: [Claire.Hanen@lip6.fr](mailto:Claire.Hanen@lip6.fr)

C. Hanen · A. Munier-Kordon  
Sorbonne Université, CNRS, LIP6, 75005 Paris, France  
e-mail: [Alix.Munier@lip6.fr](mailto:Alix.Munier@lip6.fr)

© Springer Nature Switzerland AG 2020  
W. Bożejko and G. Bocewicz (eds.), *Modelling and Performance Analysis of Cyclic Systems*, Studies in Systems, Decision and Control 241,  
[https://doi.org/10.1007/978-3-030-27652-2\\_1](https://doi.org/10.1007/978-3-030-27652-2_1)



tion scheduling as well as differences and show the main basic results of the field, following the introduction on cyclic scheduling given in [39], Chaps. 5, 6 and 8.

Dealing with data flows instead of manufacturing process means that tasks/jobs represent computation and/or data transmission. Precedence constraints are here induced by data dependencies: a job can be processed only when its input data, produced or carried by another task, is available. Notice that in a manufacturing process, a part is usually transformed, assembled, but remains in the system, although a computation task may create or delete data. Precedence constraints may also be defined when a limited memory constraint is considered. Indeed, a job  $J_i$  that has to write a data in a full memory or buffer has to wait that another one, say  $J_j$ , frees place, inducing then a precedence relation from executions of  $J_i$  to  $J_j$ . These constraints are frequently considered in embedded systems for which the overall available memory is limited.

Computations are done by physical components that, from a scheduling point of view are similar to usual processors or machines in production process though parallel processors or more complex resources from RCPSP problems are usually used [3]. However, energy saving may induce unusual constraints on the scheduling process, in particular grouping of tasks processed by the same component, in order to avoid too many on/off.

We consider in this chapter a finite set of jobs  $\mathcal{J} = \{J_i, 1 \leq i \leq n\}$  which communicate data following a Synchronous DataFlow Graph formalism. Synchronous DataFlow Graph (SDF in short) is a simple model of computation introduced by Lee and Messerschmitt [29] for the description of Digital Signal Processing Applications. In this context, SDF or extensions were considered to model H263 Encoder [9], an MP3 playback [36] or a Reed-Solomon decoder [5]. The SDF obtained do not exceed here more than eight actors. SDF associated to an application may also be generated automatically using a DataFlow language [22, 40]. The number of actors for real-life applications ranges up to 600. The size of the instances encountered in this new generation of embedded systems is significantly larger than before as they express increasingly higher levels of concurrency.

Each jobs of a fixed SDF has to be executed repeated infinitely. Thus, checking the feasibility of a SDF or evaluating its maximum throughput can be seen has a cyclic scheduling problem. One of the aim of this chapter is to investigate the relationship between cyclic scheduling problems and Dataflow problems. The main questions that these two communities have explored are the following:

- *Schedulability*: does a feasible infinite schedule exist?
- *Evaluation of the maximum throughput*: what is the structure and the performance of a schedule that maximizes the throughput?
- *Performance of a periodic schedule*: what is the optimal cycle time of a schedule with a specific periodic structure?
- *Memory optimization*: what is the minimum amount of memory to reach feasibility? or a given cycle time?

We propose in next sections, a panorama of theoretical results developed for dataflow models in connection with cyclic scheduling problems. Section 2 is ded-

icated to the presentation of the SDF model. Basic results about the precedence constraints and the normalization are recalled, leading to the definition of a feasible schedule and its normalized average cycle time. Two small examples, namely a loop parallelization problem and the modelling of periodic data transfer for a real-time system are presented in Sect. 3. Section 4 presents some basic results and optimization problems for the special case of uniform precedence graph, which is particular important class of SDFs. Section 5 is dedicated to the presentation of basic mathematical properties on SDF and two important optimization problems. Section 6 is our conclusion.

## 2 Synchronous DataFlow Graphs

This section presents some basic definitions and results on Synchronous Dataflow Graphs. Section 2 introduces the general model and the repetition vector. Next subsection recalls that a SDF models an infinite set of precedence relations between the successive executions of the jobs. Section 2.3 presents the normalization of a SDF. This transformation will be useful to study the schedulability and the determination of a periodic schedule in Sect. 5. We lastly presents some common criteria and scheduling policies.

### 2.1 General Model

Let us consider a set of  $n$  jobs  $\mathcal{J} = \{J_1, \dots, J_n\}$  with processing times  $\{p_1, \dots, p_n\}$  to be repeated many times. For  $J_i \in \mathcal{J}$ ,  $\langle J_i, k \rangle$  denotes the  $k$ th occurrence of  $J_i$ . Jobs are usually supposed to be totally or partially non-reentrant, i.e. two successive executions of a job may not overlap or for all  $n > 0$ ,  $\langle J_i, n + 1 \rangle$  starts at least one time unit after  $\langle J_i, n \rangle$  starts.

Jobs can exchange data using FIFO (First-In First Out) queues. Each FIFO has exactly one input job  $J_i$  and one output job  $J_j$  and is thus modelled by an arc  $a = (J_i, J_j)$ . Arcs are usually bi-valued by two strictly positive integers  $u(a)$  and  $v(a)$  with the assumptions that:

1.  $u(a)$  data (or tokens) are stored in the FIFO at the completion of each execution of  $J_i$ ;
2.  $v(a)$  data are removed from the FIFO before each execution of  $J_j$ . If there is not enough data, the job cannot be executed and must wait for them.

Let  $A$  be the set of arcs.  $M_0(a)$  for each arc  $a \in A$  is a non negative integer corresponding to the initial number of data in the associated buffer. These values can be fixed at the beginning, or may be variable for some optimization problems. A Synchronous DataFlow Graph (in short SDF) is then a tri-valued multi-graph  $G = (\mathcal{J}, A, u, v, M_0)$ . Let  $\mathcal{C}(G)$  denotes the set of circuit of  $G$ .

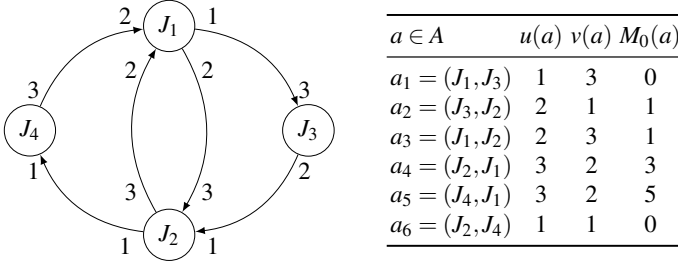


Fig. 1 A Synchronous Data Flow graph  $G$

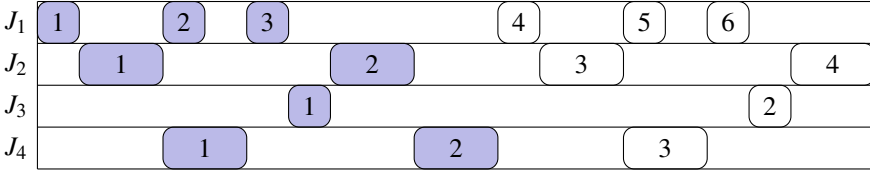


Fig. 2 First executions of the earliest schedule of the SDF pictured by Fig. 1

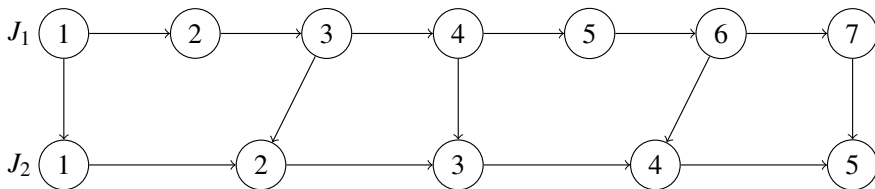
A schedule is a function  $s : \mathcal{J} \times \mathbb{N}^* \rightarrow \mathbb{R}^+$  such that  $s(J_i, k)$  is the starting time of  $\langle J_i, k \rangle$ . A schedule is feasible if at any instant the number of tokens in any FIFO is non negative.

Consider for example the SDF of  $n = 4$  jobs  $\mathcal{J} = \{J_1, J_2, J_3, J_4\}$  depicted by Fig. 1. We also suppose that  $p_1 = 1$ ,  $p_2 = 2$ ,  $p_3 = 1$  and  $p_4 = 2$ . Figure 2 presents the first executions of the earliest schedule of the SDF of Fig. 1.

Let us define the weight of any circuit  $c$  of  $G$  by  $W(c) = \prod_{a \in c} \frac{u(a)}{v(a)}$ . Note that, if  $W(c) > 1$ , the number of data items stored in the FIFO will increase as far as the jobs are executed. In the contrary, if  $W(c) < 1$ , this numbers tends to 0, leading to a deadlock. These situations correspond to design flaws and such graphs can be dismissed. Thus, all studies are restrained to **unitary graphs** for which the weight of every circuit  $c$  is 1, i.e.,  $\forall c \in \mathcal{C}(G), W(c) = 1$ .

Suppose that, at time instant  $t$ ,  $J_i$  was executed  $n_i$  times, with  $n_i > 0$  and that  $J_j$  was executed  $n_j$  times with  $n_j > 0$ . Then, the total number tokens at  $t$  stored in the buffer associated to arc  $a$  equals  $M_0(a) + u(a)n_i - v(a)n_j$ . Thus, we observe that if  $n_i = k \times v(a)$  and  $n_j = k \times u(a)$ , the number of tokens in the queue equals  $M_0(a)$ . More formally, the following theorem is proved in [29]:

**Theorem 1.1** (Repetition vector) *Suppose that  $G$  is a unitary SDF. Then, there exists an integer vector  $N \geq 1^n$  such that or for every arc  $a = (J_i, J_j) \in A$ , the equality  $u(a) \times N_i = v(a) \times N_j$  holds. Then, the graph is feasible iff each job  $J_i \in \mathcal{J}$  can be executed at least  $N_i$  times. Moreover, once each job is executed exactly  $N_i$  times (if it is possible), the systems returns in its initial state, i.e the current marking of the buffers equals its initial value.*



**Fig. 3** Precedence relations between first executions of  $J_1$  and  $J_2$  and the arc  $a = (J_1, J_2)$  with  $u(a) = 2$ ,  $v(a) = 3$  and  $M_0(a) = 1$ . Jobs  $J_1$  and  $J_2$  are supposed to be re-entrant

We can check that our example pictured by Fig. 1 is unitary. The equations verified by the repetition vectors are  $N_1 = 3N_3$ ,  $2N_3 = N_2$ ,  $2N_1 = 3N_2$ ,  $N_2 = N_4$  and  $3N_4 = 2N_1$ . The smallest integer solution is then  $N_1 = 3$ ,  $N_2 = 2$ ,  $N_3 = 1$  and  $N_4 = 2$  (Fig. 3).

## 2.2 Precedence Constraints Associated to a SDF and Useful Tokens

A precedence constraint between executions  $\langle J_i, n_i \rangle$  and  $\langle J_j, n_j \rangle$  with  $(n_i, n_j) \in \mathbb{N}^2$  expresses that  $\langle J_j, n_j \rangle$  cannot be executed before the completion of  $\langle J_i, n_i \rangle$ . Munier [34] proved that each arc  $a = (J_i, J_j)$  is equivalent to an infinite set of precedence relations between the successive executions of  $J_i$  and  $J_j$  defined the following theorem. A proof can also be found in [32].

**Theorem 1.2** (Precedence constraints associated with a FIFO queue) *Let  $J_i$  and  $J_j$  be two re-entrant jobs. A FIFO queue  $a = (J_i, J_j) \in A$  with initially  $M_0(a)$  tokens models a precedence relation between the  $n_i$ th execution of  $J_i$  and the  $n_j$ th execution of  $J_j$  iff*

$$u(a) > M_0(a) + u(a) \cdot n_i - v(a) \cdot n_j \geq \max\{u(a) - v(a), 0\}.$$

For example, let us consider the arc  $a = (J_1, J_2)$  with  $u(a) = 2$ ,  $v(a) = 3$  and  $M_0(a) = 1$ . The inequality of Theorem 1.2 becomes

$$2 > 1 + 2 \cdot n_i - 3 \cdot n_j \geq 0.$$

The couples of indexes  $(n_1, n_2)$  such that there exists a precedence relation due to  $a$  are then  $\{(1 + 3k, 1 + 2k) : k \in \mathbb{N}\}$  and  $\{(3 + 3k, 2 + 2k) : k \in \mathbb{N}\}$ .

A **useful initial marking** is such that, for any arc  $a = (J_i, J_j)$ ,  $M_0(a)$  is a multiple of  $\gcd(u(a), v(a))$ . A corollary of Theorem 1.2 is that useful initial markings are dominant [32, 33]. Moreover the initial marking  $M_0(a)$  of  $a$  may be replaced

by  $\left\lfloor \frac{M_0(a)}{\gcd(u(a), v(a))} \right\rfloor \times \gcd(u(a), v(a))$  without any influence on the precedence constraints associated to  $a$ . Thus, we only consider useful initial markings.

### 2.3 Normalization

Let us assume that a SDF  $G = (\mathcal{J}, A, u, v, M_0)$  is a unitary graph. A SDF is said to be **normalized** if there exists a positive integer vector  $Z = (Z_1, \dots, Z_n)$  such that, for any arc  $a = (J_i, J_j) \in A$ ,  $u(a) = Z_i$  and  $v(a) = Z_j$ . Marchetti and Munier [32, 33] proved the following theorem:

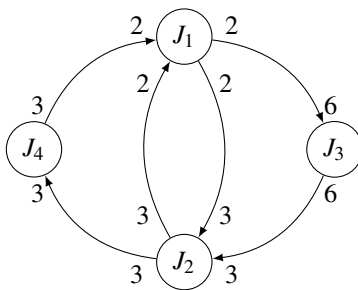
**Theorem 1.3** (Normalization) *If  $G$  is a unitary SDF then, there exists an integer vector  $Z \geq 1^n$  such that, for any arc  $a = (J_i, J_j)$ ,  $Z_i \times v(a) = Z_j \times u(a)$ . It follows that the normalized SDF  $G'$  built from  $G$  by setting, for any arc  $a = (J_i, J_j)$ ,  $u'(a) = Z_i$ ,  $v'(a) = Z_j$  and  $M'_0(a) = M_0(a) \times \frac{Z_i}{u(a)}$  generates the same set of precedence constraints as  $G$ .*

Theorem 1.3 can be seen as a corollary of Theorem 1.1. Indeed, if the repetition vector  $N$  is given, we can get the normalization vector by setting  $M = \text{lcm}(N_1, \dots, N_n)$  and for any job  $J_i$ ,  $Z_i = \frac{M}{N_i}$ . In the following we only consider normalized SDF.

For example, Fig. 4 presents the normalized SDF  $G'$  associated with the SDF  $G$  shows by Fig. 1 and its initial marking. We get  $M = \text{lcm}(2, 3) = 6$  and thus  $Z_1 = \frac{M}{3} = 2$ ,  $Z_2 = \frac{M}{2} = 3$ ,  $Z_3 = M = 6$  and  $Z_4 = \frac{M}{2} = 3$ .

### 2.4 Uniform Precedence Graphs

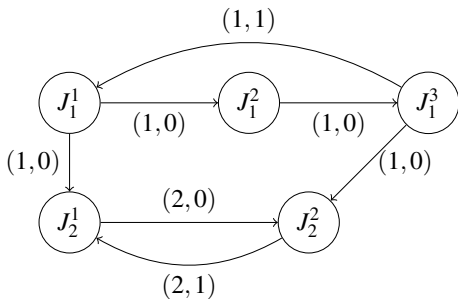
A SDF  $G$  is said to be **uniform** if for any arc  $a = (J_i, J_j)$ ,  $u(a) = v(a) = 1$ . The corresponding inequality of Theorem 1.2 becomes  $1 > M_0(p) + n_i - n_j \geq 0$ , and



$a \in A$	$M_0(a)$
$a_1 = (J_1, J_3)$	0
$a_2 = (J_3, J_2)$	3
$a_3 = (J_1, J_2)$	1
$a_4 = (J_2, J_1)$	3
$a_5 = (J_4, J_1)$	5
$a_6 = (J_2, J_4)$	0

Fig. 4 Normalized graph  $G'$  associated with  $G$

**Fig. 5** Expansion of the graph composed by two non re-reentrant jobs  $J_1$  and  $J_2$  and the arc  $a = (J_1, J_2)$  with  $u(a) = 2, v(a) = 3$  and  $M_0(a) = 1$



thus  $n_j - n_i = M_0(a)$ . In this case, the corresponding set of precedence constraints between executions of  $J_i$  and  $J_j$  verifies:

$$\forall n > 0 \quad (s(J_i, n) + p_i \leq s(J_j, n + M_0(a))).$$

Observe that in this case,  $p_i > 0$  and  $M_0(a) \geq 0$ .

However, uniform precedence graphs can be defined more generally as in [35]. Indeed, in the more general case, the two integer values associated to any arc  $a = (J_i, J_j)$  may be negative. A uniform precedence graph is then defined as a bi-valued oriented graph  $G = (\mathcal{J}, A, \ell, h)$ . The length and the height of an arc are respectively function defined as  $\ell : A \rightarrow \mathbb{Z}$  and  $h : A \rightarrow \mathbb{Z}$ . The precedence relations associated to any arc  $a = (J_i, J_j)$  are then defined by:

$$\forall n \geq \max\{1, 1 - h(a)\} \quad (s(J_i, n) + \ell(a) \leq s(J_j, n + h(a))).$$

Several authors [32, 33] have observed that the precedence relations induced by any unitary SDF can be expressed using a uniform precedence graph for which each job  $J_i$  is duplicated  $N_i$  times. This transformation, called the **expansion** of the graph, allows to consider all the algorithmic tools developed for uniform precedence graphs to SDF, and thus was extensively used.

Its main drawback is that the size of the expanded graph is not polynomial and may be huge for real-life applications. Indeed, its total number of vertices equals  $\sum_{i=1}^n N_i$  and its number of arcs is around  $\sum_{a=(J_i, J_j) \in A} \min(N_i, N_j)$ . The consequence is that the methods developed for uniform precedence graphs are not efficient for these instances. However, as we will see in Sect. 5.1, partial expansions may be considered to develop efficient exact algorithms for the throughput evaluation (Fig. 5).

## 2.5 Criteria

Several criteria may be considered to evaluate a feasible schedule  $s$ . The most common one is the average cycle time of  $s$ , which is the inverse of the throughput. More

formally, the **average cycle time** of job  $J_i$  for a schedule  $s$  is the mean time interval between two executions of  $J_i$ :

$$\lambda_i^s = \lim_{k \rightarrow +\infty} \frac{s(J_i, k)}{k}.$$

The **normalized average cycle time** of  $s$  can be defined then as

$$\lambda^s = \max_{J_i \in \mathcal{J}} \frac{\lambda_i^s}{Z_i}.$$

Another common criteria of a schedule is the **latency**  $\mathcal{L}^s$ . Roughly speaking, the latency is the maximum delay between a stimulation and the answer of the system. The SDF  $G$  must be without circuits. The latency of the entire system is the maximum time gap from a data input of a system to a connected outcome. This criteria is particularly important for real-time systems to measure the worst-case reaction time of a system.

## 2.6 Scheduling Policies

A schedule  $s$  is said to be **K-periodic** if there exists for any job  $J_i$  a period  $w_i$  and an integer  $K_i$  such that, for  $n$  sufficiently large,  $s(J_i, n + K_i) = s(J_i, n) + w_i$ .  $K_i$  is the of  $J_i$ , while  $w_i$  is its **period**. Note that

$$\lambda_i^s = \frac{w_i}{K_i}.$$

Moreover, if  $G$  is strongly connected, the normalized average cycle time is

$$\lambda^s = \frac{w_i}{K_i Z_i}.$$

The most common scheduling policy consists on executing the actors as soon as possible (asap in short) which maximizes the throughput. An asap schedule always consists of two stages [34]. The first one is an initialization phase which is necessarily finite and possibly null. A K-periodic steady state phase follows. The periodicity factor of job  $J_i$  verifies  $K_i = \alpha \times N_i$  with  $\alpha \in \mathbb{N}^*$ .

The earliest schedule depicted by Fig. 2 is K-periodic. Values  $w_i$ ,  $K_i$  and  $\lambda_i^s$  are depicted by Table 1. The normalized average cycle time equals  $\lambda^s = \frac{11}{6}$ .

The main drawback of the asap schedule is that its description is not of polynomial size. Indeed, values of the repetition vector are not polynomial and may be huge. Many authors (see as example [8, 31]) restrict their study to periodic schedules in

**Table 1** Parameters of the earliest schedule of Fig. 2

$J_i$	$w_i$	$K_i$	$\lambda_i^s$	$Z_i$
$J_1$	11	3	$\frac{11}{3}$	2
$J_2$	11	2	$\frac{11}{2}$	3
$J_3$	11	1	11	6
$J_4$	11	2	$\frac{11}{2}$	3

order to get efficient algorithms. The structure of periodic schedules of a SDF is presented in Sect. 5.2.

### 3 Modelling Examples

Two usual applications for which SDF and uniform graphs are particularly suitable are presented in this section. The first one concerns a loop parallelization. It has been studied since the early 90s [13, 20, 21, 38] and the introduction of parallel computers. Most of computation time is indeed spent in loops, so that the good use of parallelism allowed by the architecture is crucial. Our second example shows that communications between real-time periodic jobs following Liu and Layland model [30] can be expressed using a particular normalized SDF. This modelling can be used to evaluate the whole latency of the system.

#### 3.1 Loop Parallelization

Let us describe on an example how a task system associated to the execution of a loop on a specific architecture can be modeled by a uniform task system, provided that enough resources are available.

Assume that arrays  $a$ ,  $b$  are stored in the memory of a computer, and consider the C loop depicted in Fig. 6. We describe the jobs associated with assembly instructions. We assume that all instructions are processed by pipelined units, that can start a new instruction at each time unit, while the execution time till the end of an operation is 2 for additions, 6 for multiplication, and 4 for memory operations (load and store).

Figure 7 shows the uniform constraints induced by the loop semantic as well as the architecture (assuming here unlimited number of functional units). The partial reentrance is modelled by the loops around each job with label (1, 1). Although interleaving the iterations is allowed, the storage of  $a[i]$  in the memory at iteration  $i$ , i.e. job  $\langle J_9, i \rangle$ , must precede the load of  $a[i]$  at iteration  $i + 3$  (Job  $\langle J_3, i + 3 \rangle$ ). Thus the arc  $(J_9, J_3)$  has  $\ell = 4$  and  $h = 3$ . Uniform constraints can also model the use



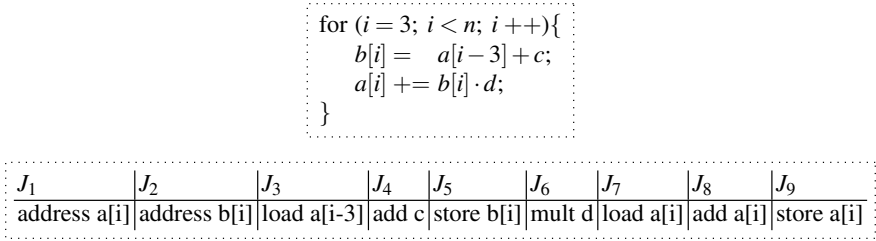


Fig. 6 A C loop and its associated jobs

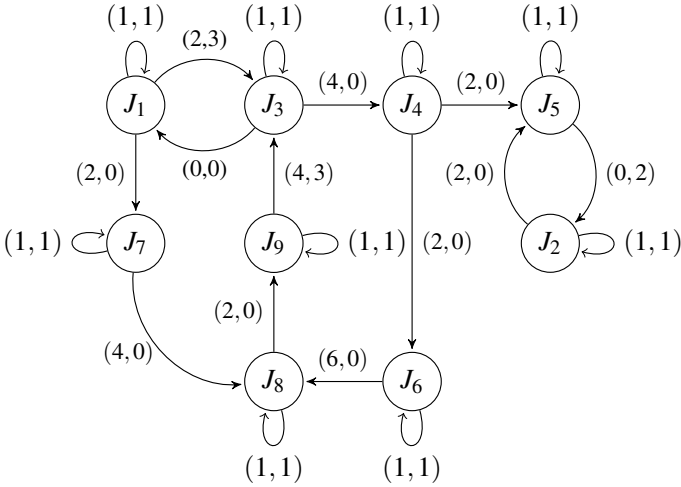


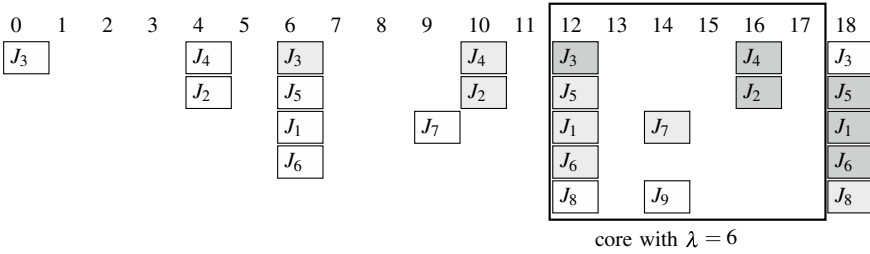
Fig. 7 A uniform graph modeling a loop. Arcs are labelled with  $(\ell, h)$

of a limited number of buffers. For example, we can assume here that the successive address of  $a[i]$  are stored in a buffer of size 3, so that at most three executions of  $J_3$  can start without starting  $J_1$  and  $J_1$  can start only if a register is free, i.e. if  $J_3$  started. This is modeled by the arcs  $(J_1, J_3)$  and  $(J_3, J_1)$  with values  $(2, 3)$  and  $(0, 0)$ .

When dealing with loop execution on parallel architectures, it is necessary to build a compact schedule, that can be easily described by a finite set of instructions. Hence in this field most authors considered strictly periodic schedules, where all jobs have the same period  $\lambda$ . Figure 8 shows an optimal periodic schedule for the graph, computed with the techniques described in Sect. 4.1.

### 3.2 Periodic Data Transfers for a Real-Time System

Let us consider a set of jobs based on the model of Liu and Layland [30]. Each job  $J_i$  is characterized by a period  $T_i$ , a processing time  $C_i$ , a deadline  $D_i$ , and a release



**Fig. 8** An optimal periodic schedule

**Table 2** Parameters of jobs  $J_1, J_2$  and  $J_3$

$J_i$	$r_i$	$T_i$	$C_i$	$D_i$
$J_1$	0	30	10	20
$J_2$	0	20	10	10
$J_3$	0	40	5	20

date  $r_i$ . The  $n$ th occurrence of  $J_i$  can be processed if and only if its execution start date  $s(t_i, n)$  is superior or equal to its release date

$$r_i + (n - 1)T_i \leq s(J_i, n).$$

and its execution end date cannot exceed its deadline

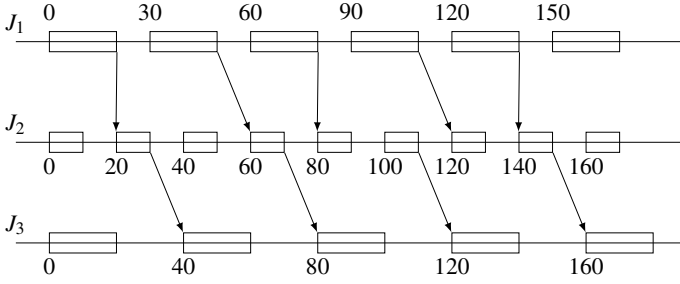
$$s(J_i, n) + C_i \leq r_i + (n - 1)T_i + D_i.$$

Suppose for example that job  $J_i$  needs data from job  $J_j$ . We consider that the  $n$ th execution of  $J_i$  writes a unique data at time  $r_i + (n - 1)T_i + D_i$  and that the  $n$ th execution of  $J_j$  reads a unique data at time  $r_j + (n - 1)T_j$ . The data are not stored in a FIFO queue, but in a unique memory. Thus, task  $J_j$  may read several time the same data if its period  $T_j < T_i$ .

For example, consider 3 jobs  $J_1, J_2$  and  $J_3$  which parameters are shown in Table 2. We assume that  $J_1$  sends data to  $J_2$  and that  $J_2$  sends data to  $J_3$ . Figure 9 presents the relations between jobs. For example,  $\langle J_2, 2 \rangle$  is reading a data from  $\langle J_1, 1 \rangle$ , while  $\langle J_2, 4 \rangle$  is reading a data from  $\langle J_1, 2 \rangle$ . The data considered by  $\langle J_2, 3 \rangle$  comes from  $\langle J_1, 1 \rangle$ , the arcs is omitted by transitivity, since  $\langle J_2, 2 \rangle$  precedes  $\langle J_2, 3 \rangle$ .

The question is to compute efficiently the latency of the system. The first problem is then to characterized couples of integers  $(n_i, n_j) \in \mathbb{N}^{*2}$  such that  $\langle J_j, n_j \rangle$  reads a data from  $\langle J_i, n_i \rangle$ .

By studying the lifetime of the data, Khatib et al. [28] observed that the relations between the executions of communicating jobs corresponds to precedence relations of a unitary SDG built following Theorem 1.4:



**Fig. 9** Communications between successive executions of jobs  $J_1$ ,  $J_2$  and  $J_3$

**Theorem 1.4** *Let  $J_i$  and  $J_j$  be two periodic jobs such that  $J_i$  communicates with  $J_j$ . The set of communicating instances of jobs  $J_i$  and  $J_j$  corresponds to precedence relations of an arc  $a = (J_i, J_j)$  of a normalized SDF with  $Z_i = T_i$ ,  $Z_j = T_j$  and  $M_0(a) = T_j + \alpha - T_a^*$  with  $T_a^* = \text{gcd}(T_i, T_j)$  and  $\alpha = \left\lceil \frac{r_i - r_j + C_i}{T_a^*} \right\rceil \times T_a^*$ .*

Thus, the corresponding SDF is composed by two arcs  $a_1 = (J_1, J_2)$  and  $a_2 = (J_2, J_3)$  with  $Z_1 = 30$ ,  $Z_2 = 20$ ,  $Z_3 = 40$  and the initial markings  $M_0(a_1) = 10$  and  $M_0(a_2) = 40$ . Note that the latency of the graph equals 60. It corresponds to the path  $\langle J_1, 3 \rangle$ ,  $\langle J_2, 5 \rangle$ ,  $\langle J_2, 6 \rangle$  and  $\langle J_3, 4 \rangle$ . Section 5.4 is dedicated to the evaluation of the latency of the SDF extracted from a real-time system.

## 4 Uniform Precedence Graphs

Some fundamental basic results on uniform precedence graphs are firstly recalled. We then introduce a generic technique, called decomposed software pipelining, that was used by several authors to solve periodic scheduling problems with resource constraints and to get approximation results. We finally present constraints recently introduced to handle energy saving in sensor networks and we mention some complexity results.

### 4.1 Basic Results

Let consider that  $G = (\mathcal{J}, A, \ell, h)$  is a uniform precedence graph  $G$ . If no additional resource constraint is considered, the schedulability, the evaluation of the maximum throughput and the performance of a periodic schedule are polynomially solvable.

These questions were initially considered for non-negative uniform case [14], i.e., for any arc  $a$ ,  $\ell(a) > 0$  and  $h(a) \geq 0$ . These results were extended in [35] for any

integer values. For the sake of simplicity, we mention here the main results for the case where  $G$  is strongly connected. General case can be found in [15, 35].

Let  $\mathcal{C}^+(G)$  (Resp.  $\mathcal{C}^-(G)$ ) be the set of circuits  $c$  of  $G$  with  $h(c) > 0$  (Resp.  $h(c) < 0$ ). For any circuit  $\mu \in \mathcal{C}(G)$ , let  $L(\mu) = \sum_{a \in c} \ell(a)$  and  $H(\mu) = \sum_{a \in c} h(a)$ . Let also define the two ratios:

$$\lambda^+(G) = \max_{\mu \in \mathcal{C}^+(G)} \frac{L(\mu)}{H(\mu)}$$

$$\lambda^-(G) = \begin{cases} \min_{\mu \in \mathcal{C}^-(G)} \frac{L(\mu)}{H(\mu)} & \text{if } \mathcal{C}^-(G) \neq \emptyset \\ +\infty & \text{otherwise} \end{cases}$$

A circuit  $\mu \in \mathcal{C}^+(G)$  is **critical** if  $\frac{L(\mu)}{H(\mu)} = \lambda^+(G)$ . The critical circuit of the graph depicted in Fig. 7 is the circuit  $\mu = (J_3, J_4, J_6, J_8, J_9, J_3)$  and its value equals  $\frac{L(\mu)}{H(\mu)} = \lambda^+(G) = \frac{18}{3} = 6$ .

A schedule  $s$  is said to be **strictly periodic** if there is a constant  $\lambda$  such that  $\forall J_i \in \mathcal{J} \forall k > 0 \ (s(J_i, k) = s_i + (k - 1)\lambda)$ .  $\lambda$  is the **average cycle time** of  $s$ , also called its **period**. First point of Theorem 1.5 deals with the schedulability. Second point concerns the evaluation of the maximum throughput while the third point is about the performance of a periodic schedule:

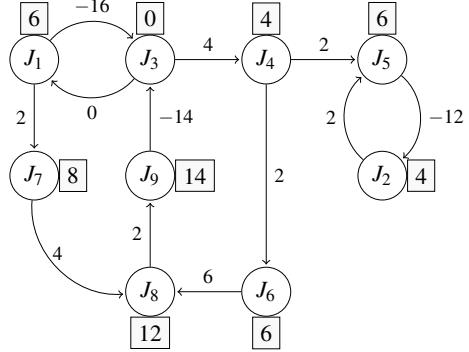
**Theorem 1.5** ([35, 39]) *Let  $G$  be a uniform strongly connected task system.*

1.  *$G$  is feasible if and only  $\lambda^+(G) \leq \lambda^-(G)$  and there is no circuit  $\mu$  in  $\mathcal{C}(G)$  with  $H(\mu) = 0$  and  $L(\mu) > 0$ .*
2. *If  $G$  is feasible, its minimum average cycle time is  $\lambda^+(G)$  and the asap schedule is  $K$ -periodic.*
3. *If  $G$  is feasible, there exists an optimal strictly periodic schedule  $s$  with  $\lambda^s = \lambda^+(G)$*
4. *Checking feasibility, computing the optimal cycle time and the optimal strictly periodic schedule can be done in polynomial time according to graph algorithms.*

Dasdan et al. [18] have experimentally tested several algorithms to compute the maximum cost to time ratio, which is exactly our problem here. Notice that a fixed value  $\lambda \in [\lambda^+(G), \lambda^-(G)]$  iff there is no valued positive cycles in the graph  $G$  valued by  $V_\lambda(a) = \ell(a) - \lambda h(a)$  for any arc  $a$ . Checking for a positive cycle in a graph can be done in polynomial time using Bellman–Ford algorithm [16]. Howard’s algorithm, which is supposed to be the most efficient for the problem, although pseudo-polynomial in the worst case, increases a lower bound  $b$  of  $\lambda^+(G)$  until the critical circuit is reached or an infeasibility is detected. Another efficient and polynomial approach is a parametric path algorithm with complexity  $O(n^4)$  [2, 27].

Figure 10 shows the graph of Fig. 7 valued by  $V_\lambda$  for  $\lambda = 6$ . First execution times  $s_i \in \mathcal{J}$  of a feasible strictly periodic schedule of period 6 are also reported.

**Fig. 10** Graph of Fig. 7 valued by  $V_\lambda$  for  $\lambda = 6$ . First execution times  $s_i \in \mathcal{J}$  of a feasible periodic schedule of period 6 are reported in the squares



## 4.2 Decomposed Software Pipelining and Resource Constrained Problems

As seen in Sect. 3.1, loop parallelization induces a uniform task system. The architecture on which the loop is executed induces additional resource constraints. From the simple case of parallel processors [24, 25] to the more complex case of RCMSP (resource-constrained modulo scheduling problem), two main approaches have been investigated, in order to find an optimal strictly periodic schedule. Although it can be easily proved that periodic schedules are not dominating schedules, their simple formulation make them very easy to implement and thus often used in loop parallelization context.

Firstly the ILP formulations, for example in [3, 19, 20] combining classical ILP formulations of resource constraints (either time-indexed or not), and linear expression of uniform constraints. In [3], several models are described and experimentally compared. The second approach, known as **decomposed software pipelining (DSP)** is based on the decomposition of the cyclic scheduling problem into two phases, **retiming** and **compaction**, the first one is related to the uniform task system, and the second to non cyclic resource constrained scheduling. In particular, several approximation algorithms have been proposed, based on this ideas [6, 10, 13, 21]. Finally, in [3], a hybrid approach combining shifting and ILP has been investigated.

In this section we describe the decomposed software pipelining technique and summarize the approximation results.

DSP relies on the notion of retiming. The main interest of this technique is to transform a set of uniform constraints into a set of usual precedence constraints, so that the remaining problem is an acyclic scheduling problem with resource constraints.

The intuition behind retiming is that while dealing with periodic schedules, the real iteration number of a job occurrence is not so important. Consider an occurrence  $\langle J_i, k \rangle$ , which corresponds to the  $(k)$ th execution of the first instance of  $J_i$ . It can also be interpreted as the  $(k + r_i)$ th execution of  $J_i(r_i)$  whose first occurrence is  $\langle J_i, -r_i \rangle$ . The height of precedence relations are then changed: if there is a

uniform constraint  $a = (J_i, J_j)$  labelled by  $(\ell(a), h(a))$ , then  $s(J_i(r_i), k) + \ell(a) \leq s(J_j(r_j), k + r_j + h(a) - r_i)$ . So the value  $r_j + h(a) - r_i$  is the height of a new uniform precedence relation between  $J_i(r_i)$  and  $J_j(r_j)$ .

**Definition 1.1** A legal retiming associates to each job  $J_i$  an integer value  $r_i$  so that:

$$r : \mathcal{J} \rightarrow \mathbb{Z}, \quad \forall a = (J_i, J_j) \in A \quad (r_j + h(a) - r_i \geq 0).$$

Now considering a legal retiming, if  $r_j + h(a) - r_i = 0$  then  $\langle J_i(r_i), k \rangle$  precedes  $\langle J_j(r_j), k \rangle$  for enough large integer  $k$ . So that the precedence relations induced by the uniform constraint is now within an iteration of the shifted jobs. Otherwise,  $\langle J_i(r_i), k \rangle$  precedes an occurrence  $\langle J_j(r_j), k' \rangle$  with  $k' > k$  which belongs to a next iteration. Hence for these new generic operations  $(J_i(r_i))_{1 \leq i \leq n}$ , the first iteration fulfills the non cyclic precedence relations given by a graph called  $G^r$  computed from  $G$  by keeping only the arcs  $a = (J_i, J_j)$  for which  $r_j + h(a) - r_i = 0$ .

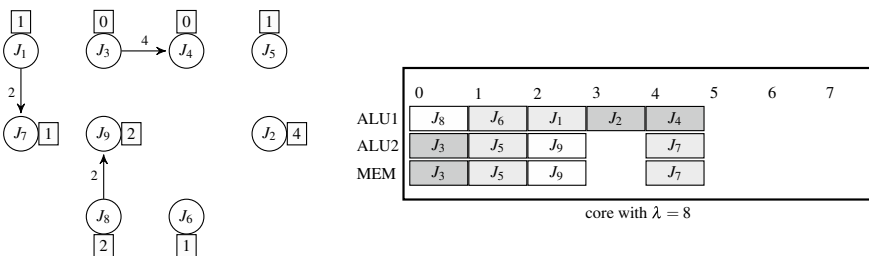
Several ideas have been investigated to find a legal retiming for nonnegative uniform task systems. Notice first that a retiming can always be found from any strict periodic schedule  $s$  fulfilling the uniform constraints.

Let  $s$  be a strict periodic schedule with period  $\lambda$ . For any job  $J_i$ ,  $s_i$  can be uniquely decomposed with respect to the period:  $s_i = t_i + \lambda \cdot q_i$ , with  $0 \leq t_i < \lambda$  and  $q_i$  is an integer.  $(t_i)_{\{J_i \in \mathcal{J}\}}$  is called the **core of the periodic schedule**, and  $(q_i)_{\{J_i \in \mathcal{J}\}}$  is the **shift of the periodic schedule**.

The shift  $(q_i)_{\{J_i \in \mathcal{J}\}}$  is a feasible retiming. This property was used by Gasperoni and Schwiegelsohn [21] by finding the shift of an optimal periodic schedule assuming unlimited resources. Figure 11 shows the graph  $G^r$  considering the retiming associated with the shift of the optimal schedule depicted in Fig. 8.

In [13], where using retiming for loop shifting is formalized, the authors consider two optimizations, with polynomial graph algorithms:

- the length of the longest path in  $G^r$  minimization
- the number of arcs in  $G^r$  minimization, so as to reduce the number of precedence constraints for loop compaction.



**Fig. 11** Retiming graph  $G^r$ , with  $r$  shown above the nodes and periodic schedule with resource constraints

The idea behind DSP approach is to choose a particular retiming  $r$ , and then use an algorithm to get a schedule  $(t_i)_{\{J_i \in \mathcal{J}\}}$  of  $G^r$ , fulfilling the resource constraints to get a periodic schedule of the original problem.

This relies on the following result:

**Theorem 1.6** *If  $r$  is a feasible retiming, and  $(t_i)_{\{J_i \in \mathcal{J}\}}$  is a schedule fulfilling the non cyclic precedence constraints of  $G^r$  and the resource constraints, then there exists a periodic schedule  $s$  whose core is  $(t_i)_{\{J_i \in \mathcal{J}\}}$  and whose shift is  $(r_i)_{\{J_i \in \mathcal{J}\}}$ .*

Figure 11 shows a construction of a core for our example, assuming that arithmetic operations are performed on one of the two available ALU's, while memory jobs (load and store) use one ALU and one memory controller at the same time. The makespan of the schedule  $(t_i)_{\{J_i \in \mathcal{J}\}}$ , combined with the observation of precedence constraints crossing the core lead to the computation of a period  $\lambda$  in polynomial time [3]. For our example  $\lambda = 8$ .

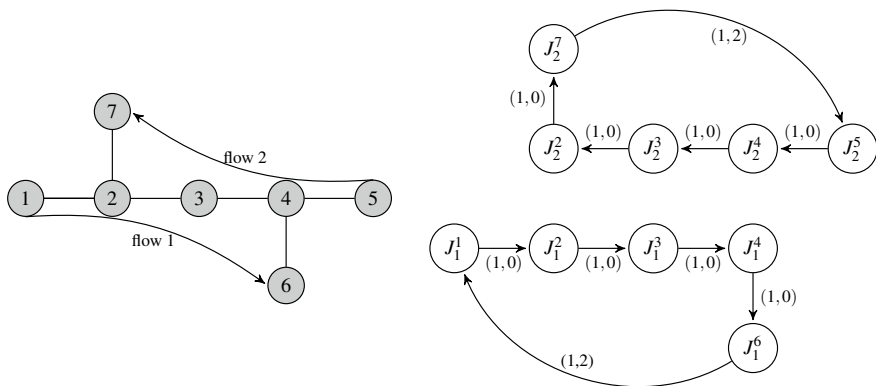
From this an interesting special case can be noted: if  $G$  has no circuit (except the ones due to the non-reentrance hypothesis for jobs), then it is always possible to get a feasible retiming  $r$  so that  $G^r$  has no arcs. Thus at the compaction step, only independent jobs have to be considered. Hence if the underlying non cyclic scheduling problem is easily solvable for independent tasks then the DSP approach provides an optimal periodic schedule. This occurs for example in cyclic shop-like problems (open shop, job-shop) if, unlike in [37], no limitation on the completion time of an iteration, or on the interleaving between iterations is given.

List scheduling algorithms are the most used heuristics for scheduling with precedence and resource constraints. Efficiency of these algorithms in practice is well known. Moreover usually a worst case performance guarantee can be determined in most resource context, from the parallel processors to RCPSP settings where a job may require several units of different resources during its execution.

Using such algorithms at the compaction step leads to a worst case ratio on the periodic schedule. This has been considered for parallel processors [17, 21] and extended to RCPSP in [6].

### 4.3 Energy Saving or Other Resource Dependent Constraints

In this section we consider problems issued from sensor networks, and in particular the scheduling problems induced by the IEEE 802.15.4/ZigBee network. Here the jobs represents data communications. Now, in real networks, while dealing with periodic schedules, the period is quite long with respect to the processing times of jobs. Moreover, the resources involved in the communication must be awoken to perform the jobs during each period. To avoid energy loss due to many in and out of the resources, it can be interesting to group jobs using the same resource as much as possible, so that the resource is awoken during a short interval once per period. In the context of periodic schedules, this will induce constraints on the core of the schedule, regardless the occurrence number of the concerned jobs.



**Fig. 12** An example of the tree  $T$  and two flows, and the associated uniform graph

Let us now present a model of data-flows inspired by the ZigBee norm, introducing grouping constraints. This work is issued from [1, 23]. We consider a tree  $T$ , whose nodes represent the clusters and whose edges represent the logical links between them. We then consider a collection of flows. Each flow  $f$  is defined by a copy of a subtree of  $T$ , oriented as an in-tree, and represents the communication of data along the communication links of  $T$  from source nodes of the flow to the unique sink. For flow  $f$ , if node  $i$  belongs to the sub-tree of  $f$ , then we denote, by  $J_f^i$ , the communication task associated to node  $i$  in flow  $f$ . Figure 12 shows an example of a tree consisting of seven clusters and two flows.

An iteration of each flow will start at each period. Moreover, the energy constraints of the ZigBee standard consider that each cluster should be active once in each period. So tasks  $J_f^i$  for all flows  $f$  passing through node  $i$  belong to group  $i$ , which should be grouped in the period.

Of course, if we do not limit the time of delivery for each flow, then the periodic scheduling problem can be handled in polynomial time by considering a retiming that lead to independent jobs. However, if we wish to achieve a good response time, we need to fix some time limits. We assume here that for each flow  $f$  the number of periods crossed by  $f$  from a source until its delivery should be less than a given integer  $p_f$ . The experiments with a scheduling tool [1], which enables system designers to configure all the required parameters of the IEEE 802.15.4/ZigBee cluster-tree WSNs, illustrate the efficiency of the model.

Moreover, this model induces for each flow a representation of the constraints induced by the data flow by a uniform graph:

1. The nodes of  $G$  are for each flow the tasks  $J_f^i$ .
2. If there is a communication link  $J_f^i \rightarrow J_f^j$  in the underlying sub-tree, then  $(J_f^i, J_f^j)$  is an arc of  $G$  with  $h$  value 0.
3. If  $J_f^i$  is the sink of the flow  $f$  and  $J_f^j$  is a source, there is an arc from  $J_f^i$  to  $J_f^j$  with  $h(J_f^i, J_f^j) = p_f + 1$ .



**Fig. 13** A core of a grouped periodic schedule—grouped jobs are shown by colors

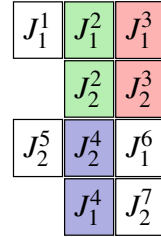


Figure 12 depicts the graph associated to the two flows, considering  $p_1 = p_2 = 1$ .

Consider now a uniform task system  $G$ , and assume that each job has a group label  $k_i \in \{1, \dots, K\}$ . A periodic schedule is said to be grouped if the tasks of the same group are executed close to each other in the core. This notion can be expressed by different means, but we can choose the simplest way here, where each group is to be scheduled as a single super-task in the core schedule. In the context we consider here, the period is usually large with respect to the processing times so we can consider that the complexity induced by the schedule of the jobs inside a super-task is not worth. As we consider here feasibility questions, we assume in the following that the super-task has a unit processing time, but the same results can be obtained by considering sum or max of the processing times of the grouped jobs.

Though the ZibBee feasibility question turns out in the following question: Given a uniform precedence graph  $G$  and group labels of the tasks, does a grouped periodic schedule of  $G$  exist? We call the *UGF* (Uniform Grouped Feasibility), this decision problem.

One can easily see that for some instances no grouped periodic schedule exists. If we consider our example assuming  $p_1 = p_2 = 0$  this means that the first execution of all jobs have to be scheduled during the first period. As  $\langle J_1^2, 1 \rangle$  precedes  $\langle J_1^3, 1 \rangle$  for the execution of the first flow, and  $\langle J_2^3, 1 \rangle$  precedes  $\langle J_2^2, 1 \rangle$  for the second flow, and as  $J_1^2, J_2^2$  (resp.  $J_1^3, J_2^3$ ) belong to the group of node 2 of the tree (resp. node 3), we get a contradiction. Figure 13 shows a core of a grouped schedule for the example of Fig. 12. We prove in [23] that the general UGF problem is NP-Complete, but the specificity of the tree underlying communication path for the ZigBee problem lead to a polynomial algorithm, based on the use of decomposed software pipelining, which proves its efficiency in practice in [1].

In [26] the authors explore a weaker way of considering grouping in a uniform task system by introducing precedence constraints with arbitrary latencies on the core schedule, called **core constraints**. Unfortunately, they prove that even if no additional resource constraints is assumed, and if unit processing times are considered, deciding the existence of a periodic schedule is also NP-complete.

## 5 Synchronous Data Flows

This section aims to present several important theoretical results on normalized SDF. The feasibility and the evaluation of the minimum normalized average cycle time are two challenging problems for which the complexity is unknown. Section 5.1 presents some algorithms to answer these two questions. Section 5.2 is dedicated to characterization of a periodic feasible schedule of minimum period, leading to a polynomial time algorithm to compute it. This characterization is considered to optimize the total buffers capacity under a minimum period constraint in Sect. 5.3. Lastly, Sect. 5.4 is dedicated to the computation of the latency for a real-time application which communications between tasks are modelled using a SDF.

### 5.1 Feasibility and Evaluation of the Minimum Normalized Average Cycle Time

Let us suppose that  $G = (\mathcal{J}, A, u, v, M_0)$  is a normalized SDF.  $G$  is feasible (or live) if there exists an infinite feasible schedule. Following Theorem 1.1, the simplest way to test the feasibility is to execute the jobs as soon as possible until each job  $J_i$  is executed at least  $N_i$  times. If it is possible,  $G$  is live.

The main drawback of this method is that values  $N_i$  are not polynomial and may be quite huge for real-life systems. From a theoretical point of view, the complexity of checking the feasibility of a SDF remains unknown. However, a simple sufficient condition of feasibility was proved by Marchetti and Munier [32, 33].

**Theorem 1.7** (Sufficient condition of feasibility of a SDF) *Let  $G$  be a normalized SDF. If, for any circuit  $c \in \mathcal{C}(G)$ , the inequality*

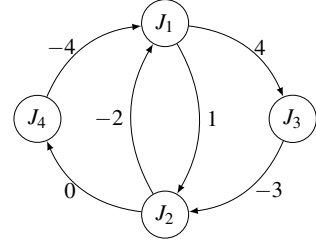
$$\sum_{a \in c} M_0(a) > \sum_{a=(J_i, J_j) \in c} (Z_j - \gcd(Z_i, Z_j))$$

*is true, then  $G$  is feasible.*

Checking this condition requires to label each arc of  $a = (J_i, J_j)$  of the SDF by  $V(a) = Z_j - \gcd(Z_i, Z_j) - M_0(a)$  and testing that the sum of the labels of each circuit remains strictly negative. As example, Fig. 14 pictures the SDF from Fig. 1 with arcs  $a = (J_i, J_j)$  valued by  $V(a)$ . This graph has no positive or null valued circuits, thus  $G$  is feasible.

Checking the existence of positive circuits can be done using a two steps polynomial time algorithm: the first step consists on checking the non existence of positive or null valued circuits using Bellman–Ford algorithm [16]. A depth-first search algorithm applied only to critical arcs allows to check the non existence of null-valued circuits.

**Fig. 14** SDF  $G$  from Fig. 1  
with arcs  $a = (J_i, J_j)$  valued  
by  $V(a) =$   
 $Z_j - \gcd(Z_i, Z_j) - M_0(a)$



Munier [34] proved that the earliest schedule of a SDF is  $K$ -periodic. Thus, the simplest way to evaluate the minimum normalized average cycle time is to compute the earliest schedule until the convergence of the normalized average cycle time. Another way is to compute the expansion of the graph, and determine its average cycle time. The main drawback of these two methods is that they are not polynomial, and thus not efficient whenever  $\sum_{i=1}^n N_i$  is important.

Bodin et al. [11] have proved that, for any integer vector of  $n$  components  $X \geq 1^n$  an expansion  $G_X$  (which is a uniform precedence graph) of  $G$  can be defined. For any arc  $a = (J_i, J_j)$ , arcs of  $G_X$  between the duplicates of  $J_i$  and  $J_j$  models a superset of precedence constraints between  $J_i$  and  $J_j$ . They also show that dominant values for the computation of the minimum normalized average cycle can be achieved for the vector set  $\{X \in \mathbb{N}^n : \forall i \in \{1, \dots, n\} \ (X_i \text{ divides } N_i)\}$ . These expansions can be used to get upper-bounds of the minimum normalized average cycle.

Algorithm 1 was also developed by Bodin et al. [12] to compute the minimum normalized average cycle time by expanding only jobs of the successive critical circuits. Although non polynomial, this algorithm allows to evaluate quickly this value for industrial instances of large size.

---

**Algorithm 1:** Computation of the minimum normalized average cycle time

---

**Require:** A normalized SDF  $G = (\mathcal{J}, A, u, v, M_0)$ .

**Ensure :** Normalized minimum average cycle time  $\lambda(G)$  of  $G$ .

- 1 Set  $M = (Z_1, \dots, Z_n)$ ,  
 $\forall i \in \{1, \dots, n\} \ (N_i = \frac{M}{Z_i})$ ;
  - 2 Set  $X = 1^n$ ,  
 $G_X$  the corresponding expanded graph,  
 $c$  a critical circuit of  $G_X$ ;
  - 3 **while** every job  $J_i$  of  $c$  is not expanded  $N_i$  times **do**
  - 4     Set  $X_i = N_i$  for every job  $J_i$  of  $c$ ;
  - 5     Update  $G_X$  and a critical circuit  $c$  of  $G_X$ ;
  - 6 Let  $\lambda(c)$  be the average cycle time of  $G_X$ ,  
 $\lambda(G) = \frac{\lambda(c)}{X_1}$ ;
-

### 5.2 Existence and Computation of a Periodic Schedule of Minimum Average Cycle Time

We show in this section that the determination of a feasible periodic schedule of minimum period is a polynomial problem for a normalized SDF. A schedule  $s$  is **periodic** if for any job  $J_i$ , there exists  $w_i \in \mathbb{Q}^{*+}$  with  $\forall n > 1$  ( $s(J_i, n) = s(J_i, 1) + (n - 1)w_i$ ). Benabid et al. [7] have proved Theorem 1.8 that characterizes periodic schedules.

**Theorem 1.8** *Let  $G$  be a normalized strongly connected SDF. For any periodic schedule  $s$ , there exists a rational  $\lambda^s \in \mathbb{Q}^{*+}$  such that for any job  $J_i$ ,  $\frac{w_i}{Z_i} = \lambda^s$ . Moreover, the precedence relations associated with any place  $a = (J_i, J_j)$  are fulfilled by  $s$  iff*

$$s(J_j, 1) - s(J_i, 1) \geq p_i + \lambda^s(Z_j - M_0(a) - \text{gcd}(Z_i, Z_j)).$$

$\lambda^s$  is then the average cycle time of  $s$ .

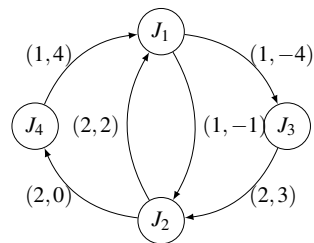
Since length  $p_i > 0$  for any job  $J_i$ , there exists a periodic schedule for  $G$  iff for any circuit  $c \in \mathcal{C}(G)$ , the inequality  $\sum_{a=(J_i, J_j) \in c} (Z_j - M_0(a) - \text{gcd}(Z_i, Z_j)) < 0$  holds, which is exactly the condition of feasibility of Theorem 1.7. If this condition is true, the minimum average cycle time  $\lambda^s$  can then be computed by finding critical circuits of the graph  $G_1$  with the same structure of  $G$  and for which each arc  $a = (J_i, J_j)$  is bi-valued by  $(p_i, M_0(a) + \text{gcd}(Z_i, Z_j) - Z_j)$ .

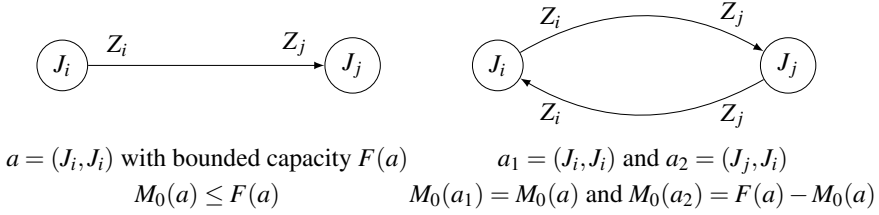
Consider for example the bi-valued graph  $G_1$  pictured by Fig. 15 and associated with the graph  $G$  of Fig. 1. The critical circuit of  $G_1$  is  $c = (J_1, J_3, J_2, J_1)$  with ratio  $\lambda(c) = \frac{1+2+2}{-4+3+2} = 5$ . Thus the minimum normalized average period of a periodic schedule is  $\lambda^s = 5$ .

### 5.3 Optimization of the Total Buffers Capacity Under a Minimum Period Constraint

SDF can be considered to model data exchanges [29] for streaming applications. Jobs correspond here to programs that are repeatedly executed. Arcs are associated to

**Fig. 15** SDF  $G$  from Fig. 1 with arcs  $a = (J_i, J_j)$  bi-valued by  $(p_i, M_0(a) + \text{gcd}(Z_i, Z_j) - Z_j)$





**Fig. 16** Transformation of an arc  $e$  with a capacity bounded by  $F(a)$  into a couple of arcs with no capacity constraint

buffers. The total amount of memory needed to execute an application is an important criteria for the designers due to the cost of the memories. Thus, the minimization of the total buffers capacity under a minimum period constraint is an important bi-criteria optimization problem.

The capacity  $F(a)$  of an arc  $a$  is the maximum number of tokens that can be stored simultaneously in the buffer corresponding to  $a$ . First at all, Marchetti and Munier proved in [31] that the capacity of a buffer may be modelled using a backward arc as follows by studying the precedence relations induced by this capacity constraints.

**Theorem 1.9** *Any arc  $a = (J_i, J_j)$  initially marked by  $M_0(a)$  with a capacity limited by  $F(a) \geq M_0(a)$  may be replaced by a couple of arcs (with no limited capacity)  $a_1 = (J_i, J_j)$  and  $a_2 = (J_j, J_i)$  with  $M_0(a_1) = M_0(a)$  and  $M_0(a_2) = F(a) - M_0(a)$  (see Fig. 16).*

Let  $G_s = (\mathcal{J}, A_s, u, v, M_0)$  be the SDF associated to  $G$  for which each arc  $a$  with a limited capacity is replaced by a couple of arcs  $(a_1, a_2)$ . For any arc  $a$ , we denote by  $\theta(a) > 0$  the size needed to store one unique data in  $a$ . The size of  $a$  is then  $F(a) \times \theta(a)$  and the total size is thus

$$\mathcal{F} = \sum_{a \in A} \theta(a) F(a) = \sum_{a \in A_s} \theta(a) M_0(a)$$

The optimization problem addressed here is to find an initial marking  $M_0(a)$ ,  $a \in A_s$  such that the total size of the memories  $\sum_{a \in A_s} \theta(a) M_0(a)$  is minimum, while there exists a schedule with a normalized average cycle time at most equal to  $K$ .

Since there is no polynomial algorithm to compute the feasibility and the minimum average cycle time of a SDF, we do not know if this problem belong to  $\mathcal{NP}$ . Several authors limit their study to periodic schedules in order to get around this problem. In this case, the optimization problem can be expressed easily using the following Integer Linear Program  $\Pi(K)$ :

minimize

$$\left( \sum_{a \in A_s} \theta(a) M_0(a) \right)$$

subject to

$$\begin{cases} \forall a = (J_i, J_j) \in A_s & (s(J_j, 1) - s(J_i, 1) \geq p_i - K(M_0(a) + \gcd(Z_i, Z_j) - Z_j)) \\ \forall a = (J_i, J_j) \in A_s & (M_0(a) = k_{i,j} \cdot \gcd(Z_i, Z_j)) \\ \forall a = (t_i, t_j) \in A_s & (k_{i,j} \in \mathbb{N}) \\ \forall J_i \in \mathcal{J} & (s(J_i, 1) \geq 0) \end{cases}$$

If the initial marking is not fixed, Marchetti and Munier proved in [31] that this problem is  $\mathcal{N}\mathcal{P}$ -complete even if  $G$  is a uniform precedence graph with  $F(a) = 1$  for every arc. Benazouz et al. [8] developed a 2-approximation ratio algorithm for the general case. The idea of this algorithm is first to solve the associated relaxed linear program. An approximated solution is then built using a classical rounding technique.

#### 5.4 Evaluation of the Latency for Real-Time Systems

Consider a normalized SDF  $G$  without circuits issued from a set of real-time Jobs which are communicating as described in Sect. 3.2. The problem is to evaluate efficiently the latency of the system. Latency is a measure of the response time of the system, it is thus fundamental for real-time systems.

Let us define the maximum (*Resp.* minimum) latency  $\mathcal{L}_{max}$  (*Resp.*  $\mathcal{L}_{min}$ ) between two connected jobs  $J_i$  and  $J_j$  as the maximum (*Resp.* minimum) duration between the end of an execution of  $\langle J_i, n_i \rangle$  and the start of an execution of  $\langle J_j, n_j \rangle$  such that there is a precedence relation from  $\langle J_i, n_i \rangle$  to  $\langle J_j, n_j \rangle$ . Theorem 1.10 proved by Khatib et al. [28], expresses the minimum and the maximum latency between two periodic communicating jobs:

**Theorem 1.10** *The maximum and the minimum latencies between a couple of periodic jobs  $(J_i, J_j)$  such that  $J_i$  communicates data to  $J_j$  are*

$$\mathcal{L}_{min}(J_i, J_j) = r_j - r_i + \alpha - C_i$$

and

$$\mathcal{L}_{max}(J_i, J_j) = r_j - r_i - \max\{0, T_i - T_j\} + \alpha - T_a^* + T_i - C_i$$

with  $T_a^* = \gcd(T_i, T_j)$  and  $\alpha = \left\lceil \frac{r_i - r_j + C_i}{T_a^*} \right\rceil \times T_a^*$ .

Consider for example the two communicating jobs  $J_1$  and  $J_2$  from Fig. 9. We get  $\alpha = 10$ ,  $T_a^* = 10$ ,  $\mathcal{L}_{min}(J_1, J_2) = 0$  and  $\mathcal{L}_{max}(J_1, J_2) = 10$ . The delay between the end of  $\langle J_1, 1 \rangle$  and the beginning of  $\langle J_2, 1 \rangle$  equals 0 and corresponds to  $\mathcal{L}_{min}(J_1, J_2)$ . The delay between the end of  $\langle J_1, 2 \rangle$  and the beginning of  $\langle J_2, 1 \rangle$  equals 10 and corresponds to  $\mathcal{L}_{max}(J_1, J_2)$ .

The latency of the entire system is a time gap from a data input of a system to a connected outcome. The worst-case latency is then the longest time gap between the input Job executions and output Job executions of a system.

The exact evaluation of the latency can be obtained by computing an expanded valued graph [28]. Each vertex  $J_i$  is duplicated  $N_i$  times. Arcs correspond to relations between executions and are valued by the exact latency between the corresponding executions following Theorem 1.10. This method is clearly not polynomial and is not efficient for large repetition vectors.

Now, let  $G_{max} = (\mathcal{J} \cup \{s, d\}, A_m, w)$  be the graph built from the SDF  $G$  as follows.

- Let  $a = (J_i, J_j)$  be an arc of  $G$  with  $T_a^* = \gcd(T_i, T_j)$ . An arc  $e = (J_i, J_j)$  is built for  $G_{max}$  with

$$w(e) = \begin{cases} \left\lceil \frac{C_i}{T_a^*} \right\rceil T_a^* + T_i - T_a^* & \text{if } T_i \leq T_j \\ \left\lceil \frac{C_j}{T_a^*} \right\rceil T_a^* + \left\lceil \frac{T_j}{T_j} \right\rceil T_j - T_a^* & \text{otherwise} \end{cases}$$

- For any job  $J_i$  without predecessor, add the arc  $e = (s, J_i)$  with  $w(e) = 0$ ; for any job  $J_i$  without successor, add the arc  $e = (J_i, p)$  with  $w(e) = D_i$ .

Khatib et al. [28] proved that an upper bound of the maximum latency can be computed by evaluating the longest paths of  $G_{max}$ . On the same way, a lower bound of the maximum latency can be computed by evaluating the longest paths of  $G_{min} = (\mathcal{J} \cup \{s, d\}, A_m, w')$  defined as follows:

- For any arc  $e \in A_m$  corresponding to an arc  $a = (J_i, J_j) \in A$ ,  $w'(e) = \left\lceil \frac{C_i}{T_a^*} \right\rceil T_a^*$ .
- For any job  $J_i$  without predecessor, add the arc  $e = (s, J_i)$  with  $w'(e) = 0$ ; For any job  $J_i$  without successor, add the arc  $e = (J_i, p)$  with  $w'(e) = D_i$

They also experimentally show that the gap between the exact value and the upper (resp. lower) bound varies between 10 and 15% (resp. 20 and 30%).

Let us consider the real-time system of Sect. 3.2 adding a communication from  $J_1$  to  $J_3$ . Corresponding graphs  $G_{min}$  and  $G_{max}$  are pictured by Fig. 17. The value of longest paths of these two graphs are respectively equal to 90 and 60.

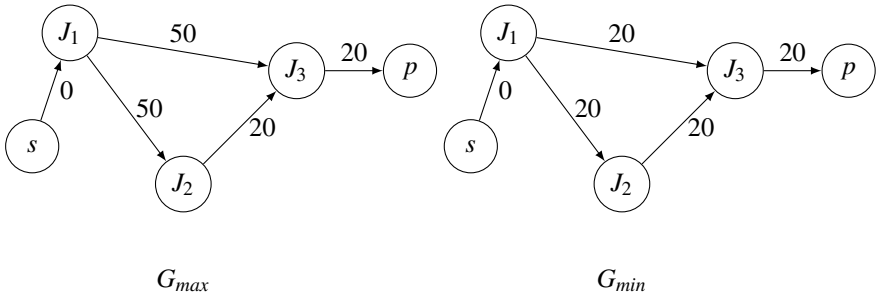


Fig. 17  $G_{min}$  and  $G_{max}$  for the real-time system of Sect. 3.2

## 6 Remarks and Conclusions

Cyclic scheduling problems arise in many crucial applications of computing and real time systems. Data transfers and data dependences induce specific precedence constraints, that have been analyzed and sometimes combined with resource constraints. In this chapter we proposed an insight on the main theoretical tools and algorithms of the field, and gave the flavor of the most recent questions and results. Several questions remain open, from complexity to algorithmic issues. We hope that the readers will further contribute to their solutions.

## References

1. Ahmad, A., Hanzálek, Z.: An energy efficient schedule for IEEE 802.15.4/zigbee cluster tree WSN with multiple collision domains and period crossing constraint. *IEEE Trans. Ind. Inform.* **14**(1), 12–23 (2018)
2. Alacaide, D., Chu, C., Kats, V., Levner, E., Sierksma, G.: Cyclic multiple robot scheduling with time-window constraints using a critical path approach. *Eur. J. Oper. Res.* **177**, 147–162 (2007)
3. Ayala, M., Benabid, A., Artigues, C., Hanen, C.: The resource-constrained modulo scheduling problem: an experimental study. *Comput. Optim. Appl.* **54**(3), 645–673 (2013)
4. Barroso, L.: The price of performance. *ACM Queue* **3**(7), 48–53 (2005)
5. Bekooij, M.J., Jansen, P.G., Smit, G.J., Wiggers, M.H.: Efficient computation of buffer capacities for cyclo-static real-time systems with back-pressure. In: 2007 44th ACM/IEEE Design Automation Conference, San Diego, CA, 4–8 June 2007, pp. 281–292. IEEE (2007)
6. Benabid, A., Hanen, C.: Worst case analysis of decomposed software pipelining for cyclic unitary RCPSP with precedence delays. *J. Sched.* **14**(5), 511–522 (2011)
7. Benabid-Najjar, A., Hanen, C., Marchetti, O., Kordon, A.M.: Periodic schedules for bounded timed weighted event graphs. *IEEE Trans. Autom. Control* **57**(5), 1222–1232 (2012)
8. Benazouz, M., Marchetti, O., Kordon, A.M., Urard, P.: A new approach for minimizing buffer capacities with throughput constraint for embedded system design. In: The 8th ACS/IEEE International Conference on Computer Systems and Applications, AICCSA 2010, Hammamet, Tunisia, 16–19 May 2010, pp. 1–8. IEEE (2010)
9. Benazouz, M., Marchetti, O., Munier-Kordon, A., Michel, T.: A new method for minimizing buffer sizes for cyclo-static dataflow graphs. In: 2010 8th IEEE Workshop on Embedded Systems for Real-Time Multimedia, Scottsdale, AZ, 2010, pp. 11–20. IEEE (2010)
10. Blachot, F., de Dinechin, B.D., Huard, G.: SCAN: a heuristic for near-optimal software pipelining. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006 Parallel Processing. Euro-Par 2006. Lecture Notes in Computer Science, vol 4128. Springer, Berlin (2006)
11. Bodin, B., Kordon, A.M., de Dinechin, B.D.: K-periodic schedules for evaluating the maximum throughput of a synchronous dataflow graph. In: 2012 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XII, Samos, Greece, 16–19 July 2012, pp. 152–159. IEEE (2012)
12. Bodin, B., Kordon, A.M., de Dinechin, B.D.: Optimal and fast throughput evaluation of CSDF. In: DAC '16 Proceedings of the 53rd Annual Design Automation Conference, Austin, Texas, 5–9 June 2016, pp. 160:1–160:6. ACM, New York (2016)
13. Calland, P.Y., Darté, A., Robert, Y.: Circuit retiming applied to decomposed software pipelining. *IEEE Trans. Parallel Distrib. Syst.* **9**(1), 24–35 (1998)
14. Carlier, J., Chrétienne, P.: Problèmes d'ordonnancement: modélisation, complexité, algorithmes. Masson, Paris (1988)




15. Claire, H.: Cyclic scheduling. In: Robert, Y., Vivien, F. (eds.) *Introduction to Scheduling*. Springer, Berlin (2009)
16. Cormen, T., Leiserson, C., Rivest, R.: *Introduction to Algorithms*. MIT Press, Cambridge (1990)
17. Darte, A., Huard, G.: Loop shifting for loop compaction. *Int. J. Parallel Program.* **28**(4/99), 415–431 (2000)
18. Dasdan, A., Irani, S., Gupta, R.K.: Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In: *DAC '99 Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, New Orleans, Louisiana, USA, 21–25 June 1999, pp. 37–42. ACM, New York (1999)
19. Dupont De Dinechin, B., Artigues, C., Azem, S.: Resource-constrained moulou scheduling. In: Artigues, C., Demasse, S., Néron, E. (eds.) *Resource Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, Control Systems, Robotics and Manufacturing Series, pp. 267–277. ISTE-Wiley (2008)
20. Eichenberger, A., Davidson, E.: Efficient formulation for optimal modulo schedulers. In: *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Las Vegas, Nevada, pp. 194–205 (1997)
21. Gasperoni, F., Schwiegelshohn, U.: Generating close to optimum loop schedules on parallel processors. *Parallel Process. Lett.* **4**, 391–403 (1994)
22. Goubier, T., Sirdey, R., Louise, S., David, V.:  $\Sigma c$ : a programming model and language for embedded manycores. In: Xiang, Y., Cuzzocrea, A., Hobbs, M., Zhou, W. (eds.) *Algorithms and Architectures for Parallel Processing — 11th International Conference, ICA3PP*, Melbourne, Australia, 24–26 October 2011, pp. 385–394. Springer, Berlin (2011)
23. Hanen, C., Hanzálek, Z.: Grouping tasks to save energy in a cyclic scheduling problem: a complexity study. *HAL CCSD* (2012)
24. Hanen, C., Munier, A.: Cyclic scheduling on parallel processors: an overview. In: Chrétienne, P., Coffman, E.G., Lenstra, J.K., Liu, Z. (eds.) *Scheduling Theory and Its Applications*. Wiley, New York (1994)
25. Hanen, C., Munier, A.: A study of the cyclic scheduling problem on parallel processors. *Discret. Appl. Math.* **57**(2–3), 167–192 (1995)
26. Hanzálek, Z., Hanen, C.: The impact of core precedences in a cyclic RCPSP with precedence delays. *J. Sched.* **18**(3), 275–284 (2015)
27. Kats, V., Levner, E.: Cyclic routing algorithms in graphs: performance analysis and applications to robot scheduling. *Comput. Ind. Eng.* **61**(2), 279–288 (2011)
28. Khatib, J., Kordon, A.M., Klikpo, E.C., Trabelsi-Colibet, K.: Computing latency of a real-time system modeled by synchronous dataflow graph. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016*, Brest, France, 19–21 October 2016, pp. 87–96. ACM, New York (2016)
29. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. *Proc. IEEE* **75**(9), 1235–1245 (1987)
30. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **20**(1), 46–61 (1973)
31. Marchetti, O., Kordon, A.M.: Complexity results for weighted timed event graphs. *Discret. Optim.* **7**(3), 166–180 (2010)
32. Marchetti, O., Munier-Kordon, A.: Cyclic scheduling for the synthesis of embedded systems. In: Robert, Y., Vivien, F. (eds.) *Introduction to Scheduling*. Springer, Berlin (2009)
33. Marchetti, O., Munier-Kordon, A.: A sufficient condition for the liveness of weighted event graphs. *Eur. J. Oper. Res.* **197**(2), 532–540 (2009)
34. Munier, A.: Régime asymptotique optimal d'un graphe d'événements temporisé généralisé: application à un problème d'assemblage. *RAIRO-Automatique Productive Informatique Industrielle* **27**(5), 487–513 (1993)
35. Munier-Kordon, A.: A graph-based analysis of the cyclic scheduling problem with time constraints: schedulability and periodicity of the earliest schedule. *J. Sched.* **14**(1), 103–117 (2011)
36. Oh, H., Ha, S.: Efficient code synthesis from extended dataflow graphs for multimedia applications. In: *Proceedings of the 39th Design Automation Conference, DAC 2002*, New Orleans, LA, USA, 10–14 June 2002, pp. 275–280. ACM, New York (2002)

37. Pempera, J., Smutnicki, C.: Open shop cyclic scheduling. *Eur. J. Oper. Res.* **269**(2), 773–781 (2018)
38. Rau, B.R.: Iterative modulo scheduling: an algorithm for software pipelining loops. In: *MICRO 27: Proceedings of the 27th Annual International Symposium on Microarchitecture*, 30 November–2 December 1994, pp. 63–74. ACM, New York (1994)
39. Robert, Y., Vivien, F.: *Introduction to Scheduling*. Chapman and Hall/CRC Press, Boca Raton (2009)
40. Thies, W., Karczmarek, M., Gordon, M., Maze, D.Z., Wong, J., Hoffman, H., Brown, M., Amarasinghe, S.: *StreamIt: a compiler for streaming applications*. Technical report, Massachusetts Institute of Technology (2001)

# Cyclic Two Machine Flow Shop with Disjoint Sequence-Dependent Setups



Wojciech Bożejko , Czesław Smutnicki , Mariusz Uchroński   
and Mieczysław Wodecki 

**Abstract** The chapter considers the problem of cyclical jobs scheduling on two machines with resource constraints often encountered in practice, and concerning a number of teams that can perform setups of machines between jobs performed. We are considering a fundamental and most restrictive case with only one setup team. This limitation significantly impedes the considered issue because the solution is represented here not only by the order of performing jobs, but also by the route of the setup team, i.e. the order in which the team makes setups of machines.

## 1 Introduction

The scheduling problems with sequence-dependent setups are closely related to solving the traveling salesman problem (TSP). Therefore, the solution's properties and methods of solving for the traveling salesman problem can be easily adapted and used to develop methods for solving multi-machine problems with sequence-dependent setup times. Corwin and Esogbue [5] for the two-machine problem and Bellman [3]

---

W. Bożejko (✉) · M. Uchroński  
Department of Control Systems and Mechatronics, Faculty of Electronics,  
Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St.,  
50-372 Wrocław, Poland  
e-mail: [wojciech.bozejko@pwr.edu.pl](mailto:wojciech.bozejko@pwr.edu.pl)

M. Uchroński  
e-mail: [mariusz.uchronski@pwr.edu.pl](mailto:mariusz.uchronski@pwr.edu.pl)

C. Smutnicki  
Department of Computer Engineering, Faculty of Electronics, Wrocław University  
of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland  
e-mail: [czeslaw.smutnicki@pwr.edu.pl](mailto:czeslaw.smutnicki@pwr.edu.pl)

M. Wodecki  
Department of Telecommunications and Teleinformatics, Faculty of Electronics,  
Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St.,  
50-372 Wrocław, Poland  
e-mail: [mieczyslaw.wodecki@pwr.edu.pl](mailto:mieczyslaw.wodecki@pwr.edu.pl)

as well as Held and Karp [9] for TSP propose using dynamic programming (TSP equivalent to single machine scheduling problem with sequence-dependent setups and makespan criterion  $C_{\max}$ ). The problem is strongly NP-hard because it constitutes a generalization on two machines of a single-machine cyclic scheduling problem with setups, which in turn is the same in terms of the optimized minimum cycle time with a single-machine non-cyclical problem with the  $C_{\max}$  criterion. Mentioned methods can work with problems of the size  $n \leq 15$ , which is very small. Problems with a relative small number of tasks can also be solved exactly by the branch and bound algorithms (B&B, Gupta [7], Gupta and Darrow [8]). However, the calculation times required for dynamic programming algorithms and B&B are too long, even for moderate-size problems. In turn, comparison of heuristic methods can be found in the review paper of Ruiz and Maroto [10], also in parallel versions, Taillard [12], Bożejko [1], Bożejko et al. [4] and with the use of special properties of the problem, Bożejko [2], Bożejko et al. [3].

## 2 Problem Description

The two machine flow shop problem under consideration is generated by practice and can be formally defined in the following way. We have a set of jobs

$$\mathcal{J} = \{1, 2, \dots, n\}, \quad (1)$$

which should be executed cyclically on machines from the set

$$\mathcal{M} = \{1, 2\}. \quad (2)$$

Each job  $i \in \mathcal{J}$  consists of two operations, which are executed on a machine by the time  $p_{i,j}$ ,  $j = 1, 2$ . Additionally, a sequence-dependent setup should be done between operations  $i, k \in \mathcal{J}$  on a machine  $j \in \mathcal{M}$  in time  $s_{ik}^j$ ,  $j = 1, 2$ . Additionally, there exist only one setup team, which service both machines.

During the execution of tasks, the following technological and sequence constraints must be met:

- (a) every operation must be performed without interrupting on a dedicated machine,
- (b) only one task can be performed on the machine at a time,
- (c) between the operations on the machines, there must be setups performed,
- (d) the task can start on the second machine if it is terminated on the first machine,
- (e) at any time the setup can be performed on one machine only,
- (f) each operation and setup is executed sequentially (in successive so-called MPSes) after the cycle time is completed.

A set of jobs of a single cycle is called an MPS (*minimal part set*). MPSes are processed cyclically, one by one. We index successive MPSes by  $x = 1, 2, \dots$ . The problem consists in finding a schedule, which minimizes the cycle time criterion.

## 2.1 Solution Representation

The schedule in  $x$ th MPS is defined fundamentally by two matrices, namely  $(S^x, P^x)$ , with matrix elements being the real numbers. Precisely,  $S^x = [S_{i,j}^x]_{m \times n}$ , where  $S_{i,j}^x$  denotes the beginning time moment of execution of job  $j$  on  $i$ th machine in  $x$ th MPS,  $j = 1, 2, \dots, n, i = 1, 2, x = 1, 2, \dots$ . Similarly,  $P^x = [P_{i,j}^x]_{m \times n}$ , where  $P_{i,j}^x$  denotes the beginning time moment of setup operation after performing job  $j$  on  $i$ -th machine in  $x$ -th MPS,  $j = 1, 2, \dots, n, i = 1, 2, x = 1, 2, \dots$ . We assume that jobs in following MPSes are performed cyclically, which implies that exists a constant  $T$ , called cycle time, satisfying some preceding constraints, such that

$$S_{i,j}^{x+1} = S_{i,j}^x + T, \quad P_{i,j}^{x+1} = P_{i,j}^x + T, \quad i = 1, 2, j = 1, \dots, n, x = 1, 2, \dots \quad (3)$$

Notice, due to regular replication of MPSes (3), there is enough to find the schedule  $(S, P)$  for any fixed  $x$  (e.g.  $x = 1$ ) and make its shift by  $x \cdot T$ ,  $x = 1, 2, \dots$  on timeline.

Operating on schedule represented by  $(S, P)$  appears rather inconvenient for most metaheuristic approaches (local search, evolutionary search, etc.) because of troubles with potential schedule modification. Therefore, we propose to represent the solution by an equivalent combinatorial object(s), showing only the suitable transformation from this object to the schedule with their particular properties.

Regarding to  $S$ , processing order of jobs in single MPS is represented by a permutation  $\pi = (\pi(1), \pi(2), \dots, \pi(n))$  on the set  $\mathcal{J}$ . The set of all permutations on  $\mathcal{J}$  we denote by  $\Pi_n$ . From the definition, jobs in each MPS and jobs on each machine are executed in the same order. Let us assume for a moment that  $\pi$  is fixed. Since we have single setup team, then the processing order regarding to  $P$  can be determined by a sequence  $\omega = (\omega(1), \omega(2), \dots, \omega(o))$ ,  $o = 2n$ , with some additional constraints, where  $\omega(i) = (\sigma(i), \tau(i) + 1)$ . Additional constraints are as follows. The sequence  $\omega$  has to contain exactly two elements with the same  $\sigma(i)$  for some  $i$ , both of them must differ by  $\tau(i)$ ,  $\sigma(i) \in \mathcal{J}$ ,  $\tau(i) \in \{0, 1\}$ . Element  $\omega(j) = (\sigma(j), \tau(j) + 1)$  means that setup team realizes setup immediately after finishing job  $\sigma(j)$  on machine 1 if  $\tau(j) = 0$  and on after finishing the same job on machine 2 if  $\tau(j) = 1$ . In the Sect. 2.3 we will show that  $\omega$  can be completely represented by a binary sequence  $\tau = (\tau(1), \tau(2), \dots, \tau(o))$  with some additional properties.

Cycle time  $T$  depends of course on  $\pi$  and  $\omega$ . The minimum its value for a fixed  $\pi$  and  $\omega$ , will be called *minimum cycle time* and denoted by  $T(\delta)$ ,  $\delta = (\pi, \omega)$ . One asks about the basic method of transforming combinatorial representation of the solution  $\delta = (\pi, \omega)$  into cyclical schedule  $(S, P)$ . Increasing of beginning time moment of a job execution by multiples of the cycle time, we obtain the beginning moments of jobs execution in any of MPSes (commencement of execution of any of the operation in the consecutive MPS should be increased by the cycle time). Let  $\Phi = \Pi_n \times \Omega_o$  be a set of solutions representations, where  $\Pi_n$  is the set of all permutations of the  $n$ -element set  $\mathcal{J}$ , and  $\Omega_o$  is the set of all binary sequences with the length of  $o$

containing exactly  $n$  zeros and  $n$  ones. Therefore, the considered in the work problem comes down to the determination of solutions  $(\pi, \omega) \in \Phi$  which minimize the value of the minimal cycle time  $T(\delta)$ ,  $\delta = (\pi, \omega)$ .

## 2.2 Representation of the Route of the Setup Team

The route of the setup team  $\tau$  represented as binary sequence is an interesting mathematical object. Not all the sequences containing  $n$  zeros and  $n$  ones generate a feasible solution. It is a sequence in which to the position  $i$ th (counting from the left) there must be situated at most one more than the number of zeroes to the left from this position. This is due to the fact that the second machine can be setup for a task that has not yet been done on the first machine, however, it is not possible to setup the machine “two tasks ahead”—the second machine from the task that has not yet begun on the first machine cannot be setup. These sequences for small values of  $n$  can be presented in the following way:

$n = 1$ : 2 sequences {01, 10},

$n = 2$ : 5 sequences {0011, 0101, 0110, 10011010},

$n = 3$ : 14 sequences {000111, 001011, 001101, 001110, 010011, 010101, 010110, 011001, 011010, 100011, 100101, 100110, 101001, 101010}.

Catalan numbers, on the other hand, are expressed in the sequence 2, 5, 14, 42, ... and the general expression can be represented as

$$\binom{(2n)!}{n!(n+1)!}.$$

This leads us to the following lemma.

**Lemma 2.1** *The number of feasible routes of the setup team for  $n$  tasks is equal to  $n$ th Catalan number.*

*Proof (Transformation to parentheses placement problem)* The routes of the setup team can be represented by binary sequences with a length of  $2n$ , with  $n$  zeros and  $n$  ones. These sequences are characterized by the feature that to the  $i$ th position,  $i = 1, 2, \dots, 2n$ , counting from the left, there can appear at most one more one than the number of zeros. It turns out that the number of acceptable sequences corresponds to the number of placement of parentheses: open (“0”) ones and closed (“1”) ones—before each of the considered sequences it is enough to add at the beginning—an artificial “zero” and at the end—an artificial “one”. For example, the allowable sequences representing the routes of the setup team for  $n = 2$  are {01, 10}. After adding the artificial zero at the beginning and one at the end, we get the sequences {0011, 0101} corresponding to correctly placed parentheses: (()) and ()(). In turn, the fact of representing the number of expressions containing  $n$  pairs of parentheses

which are correctly matched as Catalan numbers is commonly known (see Graham, Knuth, Patashnik [6]). ■

### 2.3 Mathematical Model

For a fixed order of execution of jobs  $\pi$  and setup team route  $\omega$  (discussed further in detail), optimum value of cycle time  $T(\delta)$ , can be determined by solving a linear programming (LP) task. Since LP belongs to P-class in the sense of computational complexity, this approach implies the first polynomial time algorithm of finding  $T(\delta)$  and corresponding schedule  $(S, P)$  for the given  $\delta$ .

To make the required transformation, we expressed listed the above constraints with respect to single MPS (called any but common  $x$ ), allowing us to skip superscript in  $(S, P)$ , in the form LP model:

$$\min_{S, P, T} T = T(\delta) \quad (4)$$

s.t.

$$S_{i, \pi(j)} \geq P_{i, \pi(j-1)} + s_{\pi(j-1), \pi(j)}^i, \quad j = 1, 2, \dots, n, \quad i \in \mathcal{M}, \quad (5)$$

$$S_{i, \pi(1)} + T \geq P_{i, \pi(n)} + s_{\pi(n), \pi(1)}^i, \quad i \in \mathcal{M}, \quad (6)$$

$$P_{i, j} \geq S_{i, j} + p_{i, j}, \quad j = 1, 2, \dots, n, \quad i \in \mathcal{M}, \quad (7)$$

$$S_{2, j} \geq S_{1, j} + p_{1, j}, \quad j \in \mathcal{J}, \quad (8)$$

$$P_{\tau(j), \sigma(j)} \geq P_{\tau(j-1), \sigma(j-1)} + s_{\sigma(j-1), \sigma(j)}^{\tau(j)}, \quad j = 1, 2, \dots, o, \quad (9)$$

$$P_{\tau(1), \sigma(1)} + T \geq P_{\tau(o), \sigma(o)} + s_{\sigma(o), \sigma(1)}^{\tau(o)}, \quad j = 1, 2, \dots, o, \quad (10)$$

where  $\delta = (\pi, \omega)$ , and  $\omega = (\omega(1), \dots, \omega(o))$ ,  $\omega(j) = (\tau(j), \sigma(j))$  is the sequence of the given property, i.e. fulfilling equality

$$\sigma(j) = \begin{cases} \sum_{k=1}^{\pi(j)} \bar{\tau}_j & \text{if } \tau_j = 0, \\ \sum_{k=1}^{\pi(j)} \tau_j & \text{if } \tau_j = 1, \end{cases} \quad (11)$$

where  $\bar{\tau}_j$  means binary negation of the bit  $\tau_j$ . Describing Eq. (9) one can observe, that the setup should be made between a job  $\sigma(j-1)$  and its direct machine successor, denoted by  $Succ(\sigma(j-1))$ . Let us assume, that

$$\sigma(j-1) = \pi(z), \quad (12)$$

for a certain  $z \in \{1, 2, \dots, n\}$ . This  $z$  can be calculated from the inverse permutation to  $\pi$ , denoted by  $\pi^{-1}$ :

$$z = \pi^{-1}(\sigma(j-1)). \quad (13)$$

Next, we need a successor of  $\pi(z)$  which is  $\pi(z+1)$ . Inserting (13) in place of  $z$  we obtain the number of successor of  $\sigma(j-1)$  as

$$Succ(\sigma(j-1)) = \pi(z+1) = \pi(\pi^{-1}(\sigma(j-1)) + 1) \quad (14)$$

and that is why such an index is used in the index of the setup

$$s_{\sigma(j-1), Succ(\sigma(j-1))}^{\tau(j-1)+1} = s_{\sigma(j-1), \pi(\pi^{-1}(\sigma(j-1))+1)}^{\tau(j-1)+1} \quad (15)$$

in the Eq. (9).

Referring the above model to the problem limitations (a)–(f) we can describe the following relationships. Let  $C_{i,\pi(j)}$  represents the time of finishing of the job  $\pi(j)$  on the machine  $i$  (it is not needed in the LP model, but is useful for explanation). The constraint (a) can be written as

$$C_{i,\pi(j)} = S_{i,\pi(j)} + p_{i,\pi(j)} \quad i \in \mathcal{M}, \quad j \in \mathcal{J}. \quad (16)$$

It is represented in LP model in the Eq. (7), because the setup after a job must not begin before  $S_{i,j} + p_{i,j}$  which is  $C_{i,j}$  if there are no breaks during jobs execution.

To obtain a feasible solution of the problem, the following inequality must be also met:

$$S_{i,\pi(j)} \geq C_{i,\pi(j-1)} + s_{\pi(j-1), \pi(j)}^i \quad (17)$$

meaning the need to perform setups between operations (constraint (c)), and at the same time constraint (b) prohibiting the performance of two operations simultaneously on the same machine. This constraint is represented in LP model by the Eq. (5). Further

$$S_{2,\pi(j)} \geq C_{1,\pi(j)}, \quad j \in \mathcal{J} \quad (18)$$

representing sequencing constraint (d) is modeled by the Eq. (8) in the LP model.

Equation (9) refers to the equation (e) of execution a setup on at most one machine at any point of time.

In turn condition (f) for jobs cyclicity can be presented in the form of

$$C_{i,\pi(n)}^x + s_{\pi(n), \pi(1)}^i \leq S_{i,\pi(1)}^{x+1} \quad i = 1, 2, \quad (19)$$

so taking advantage of (3)

$$S_{i,\pi(n)}^x + p_{i,\pi(n)} + s_{\pi(n), \pi(1)}^i \leq S_{i,\pi(1)}^x + T \quad i = 1, 2. \quad (20)$$



for jobs, which can be derived from Eqs. (5) and (7) and assuming  $x = 1$  (the number of MPS which is considered does not matter). Equation (10) gives the setups cyclicity condition.

We are considering the goal function of the minimal cycle time  $T(\delta) = \min_{T \in \mathbb{R}} T$  fulfilling all the above constraints for the solution  $\delta = (\pi, \omega)$  (representing the order of executing of  $\pi$  tasks and the route of the setup team  $\omega$ ). The problem is to find a solution that minimizes the minimal cycle time, i.e. we are looking such  $\delta^* = (\pi^*, \omega^*)$ , that

$$T(\delta^*) = \min_{\delta \in \Phi} T(\delta). \quad (21)$$

## 2.4 Representation of the Solution

Let

$$\mathcal{S} = (S_{i,j}), \quad j = 1, 2, \dots, n, \quad i = 1, 2, \quad (22)$$

$$\mathcal{P} = (P_{i,j}), \quad j = 1, 2, \dots, n, \quad i = 1, 2, \quad (23)$$

$$\delta = (\pi, \omega), \quad \omega = (\omega(1), \dots, \omega(o)), \quad \omega(j) = (\tau(j), \sigma(j)), \quad o = 2n, \quad (24)$$

be sequences of respectively: moments of starting operations on machines  $\mathcal{S}$ , starting times of setups performed by the team setting up  $\mathcal{P} = (P_{i,j})$ ,  $i = 1, 2$ ,  $j = 1, 2, \dots, n$  and the route of the setup team represented by the sequence  $\omega$  consisted of the sequence of jobs  $\sigma$  directly after which a setup is performed and the binary sequence  $\tau$  in which 0 denotes visiting the machine 1 and 1 visiting machine 2.

The solution to the considered problem is one of three objects:

1.  $\Theta = (\mathcal{S}, \mathcal{P})$ —pair of jobs and setups beginning time moments,
2.  $\delta = (\pi, \omega)$ —jobs and setup routes, where  $\pi$  is permutation of  $n$ -element sequence, and  $\omega$  represents setup team route, wherein  $\tau$  is a binary sequence,
3. graph  $G$ —as a network representation of the researched model.

In the further part of the chapter the graph  $G$  and the pair  $(\pi, \omega)$  will be considered.

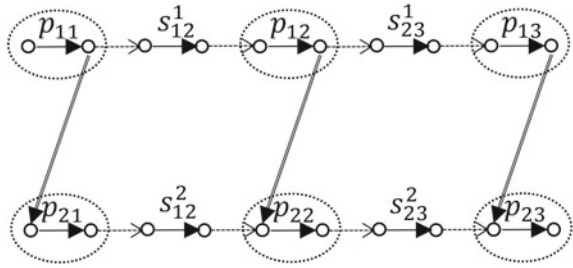
## 3 Graph Model

Let  $G(\delta) = (V, E)$  for solution  $\delta = (\pi, \omega)$  be a weighted graph with weight function  $w : E \rightarrow \mathbb{R}$ . Next, let the set of vertices  $V = \{1, 2, \dots, 8n\}$ , and a set of arcs

$$E = E_o \cup E_s \cup E_\pi \cup E_\tau \cup E_v, \quad (25)$$

where the individual sets of arcs represent:

**Fig. 1** Skeleton of the graph  $G(\delta)$



$E_o$  operations; arcs from  $E_o$  are burdened with the value of the duration of  $i$ th operation on the machine  $j$ , i.e.  $p_{ij}$ ,

$E_s$  setups; arcs from  $E_s$  burdened with the value of the duration of setup from operation  $i$  to operation  $k$  on machine  $j$ , i.e.  $s_{i,k}^j$ ,

$E_\pi$  order of execution on the machine (dependent on  $\pi$ ),

$E_\tau$  route of setup team (dependent on  $\tau$ ),

$E_v$  technological order, i.e. the route of tasks between machines 1 and 2.

The procedure for constructing a graph for a given solution with the specified order  $\pi$  and the route of the setup team  $\sigma$  can be described as follows:

1. construct a “skeleton” based on the order  $\pi$  using all vertices from the set of  $V$  and arcs from the sets of  $E_o$  (operations),  $E_s$  (setups),  $E_\pi$  (order on the machine) and  $E_v$  (technological order), as in Fig. 1.
2. add the route arcs of the setup team based on the sequence  $\tau$  according to the following idea. Let

$$n_i^1 = \sum_{j=1}^i \bar{\tau}_j \quad (26)$$

will be the counter as the  $i$ th setup is performed on the machine 1, and

$$n_i^2 = \sum_{j=1}^i \tau_j \quad (27)$$

will be the counter as the  $i$ th setup is performed on machine 2 (see Fig. 2).

The procedure of adding arcs from the set  $E_\tau$  to skeletal graph is presented by Algorithm 1. Horizontal setup arcs are in fact not needed in the algorithm analysis, so we will skip them in further considerations (the longest paths always goes through operations arcs with the weight  $p_{i,k}$  instead of horizontal setup arcs (lines 6 and 9 of the algorithm)).

The result of the procedure presented in the form of the Algorithm 1 on the graph from Fig. 1 is the graph presented in Fig. 2. Its computational complexity is  $O(n)$  and determines the complexity of the sequential method of constructing of a graph if it

---

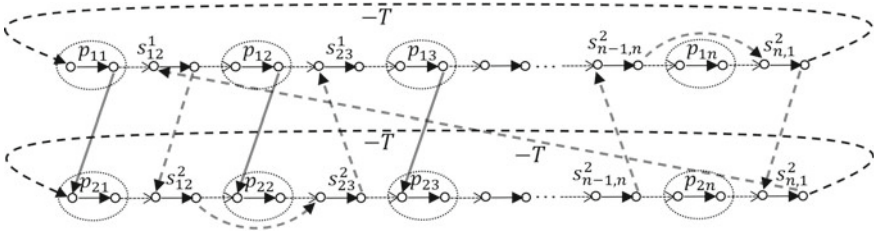
**Algorithm 1:** GraphMake( $\delta, n, n^1, n^2$ ) function

---

**Input :** solution  $\delta = (\pi, \tau)$ , variables  $n, n^1, n^2$ ;

**Output:** graph  $G(\delta)$ ;

- 1 Prepare set of vertices  $V$  and part of the set of edges (skeleton)  
 $E = E_o \cup E_s \cup E_\pi \cup E_v$  which are not dependent on  $\tau$ , as described in Sect. 3 and shown in Fig. 1.
  - 2  $E_\tau \leftarrow \emptyset$ ;
  - 3 **for**  $i = 2, 3, \dots, 2n$  **do**
  - 4 Consider elements  $(\tau_{j-1} \tau_j)$
  - 5 **if**  $(0\ 0)$  **then**
  - 6 | **null**;
  - 7 **if**  $(1\ 1)$  **then**
  - 8 | **null**;
  - 9 **if**  $(0\ 1)$  **then**
  - 10 | Insert an arc to  $E_\tau$  between setup  $n_{j-1}^1$  on the 1st machine and  $n_j^2$  on the 2nd machine
  - 11 **if**  $(1\ 0)$  **then**
  - 12 | Insert an arc to  $E_\tau$  between setup  $n_{j-1}^2$  on the 2nd machine and  $n_j^1$  on the 1st machine
  - 13 Prepare cyclic arcs for setup team: insert an arc to  $E_\tau$  between the last setup on machine  $(\tau_{2n} + 1)$  and the first setup on machine  $(\tau_1 + 1)$  with the weight  $(-T)$
  - 14  $E \leftarrow E \cup E_\tau$ ;
  - 15 Prepare cyclic arcs for jobs:
  - 16 **for**  $i = 1, 2$  **do**
  - 17 | Insert an arc to  $E$  between the last job executed on machine  $i$  and the first job on machine  $i$  with the weight  $(-T)$
  - 18 **return**  $G = (V, E)$
- 



**Fig. 2** Graph  $G(\delta)$  with cyclic arcs

is remembered in the form of a list of neighborhoods. When the graph is represented as a neighborhood matrix, the complexity increases to  $O(n^2)$  due to operating on the square matrix  $n \times n$ .

*Remark 2.1* The problem of minimal cycle time determination  $T(\delta)$  comes down to determine such a  $T$ , that the maximal length of a cycle in the graph  $G(\delta)$  is 0.

## 4 Properties

Taking into account the above assumptions of the model, the following statement can be formulated.

**Theorem 2.1** *For the given solution  $\delta = (\pi, \omega)$ ,  $\omega = (\omega(1), \dots, \omega(o))$ ,  $\omega(j) = (\tau(j), \sigma(j))$ , if the binary sequence  $\tau$  remembered in  $\omega$  representing the route of the setup team has to some  $i$ th position by one more 1 than number of 0, i.e. the setup team visits the second machine  $a$  by one greater number times the than the first machine, then there exists such a binary sequence  $\tau'$  representing the route of the setup team in which to the  $i$ th position there are as many 0 as 1 and the value of the goal function of  $\delta' = (\pi, \omega')$ ,  $\omega' = (\omega'(1), \dots, \omega'(o))$ ,  $\omega'(j) = (\tau'(j), \sigma(j))$ , meets inequality:*

$$T(\delta') \leq T(\delta). \quad (28)$$

*Proof* Let  $a, b, c, d$  be earliest time moments of events represented by appropriate vertices in  $G(\delta)$ , as shown on Gantt chart (Fig. 3), represented by a graph shown in Fig. 4. For the simplification, the following notion will be used:

$$\begin{aligned} s_{i,i+1}^1 &\longrightarrow s^1, \\ s_{i+1,i+2}^2 &\longrightarrow s^2, \\ p_{1,i+1} &\longrightarrow p^1, \\ p_{2,i+1} &\longrightarrow p^2, \\ p_{2,i+2} &\longrightarrow p^3. \end{aligned}$$

Let us approximate  $c$  and  $d$  values before and after the move which considers in swapping subsequence  $*10*$  into  $*01*$  of  $\tau$  (where  $*$  means any sequence of binary digits in which there is the same number of 0 as 1), as shown in Fig. 5. Values after the move, i.e. removing an arc presented by solid arrow between setups  $s^1$  and  $s^2$  and an arc represented by dashed one we will denote by  $c'$  and  $d'$ . If we prove, that  $c' \leq c$  and  $d' \leq d$  it will prove, that the move does not increase value of the goal function  $T$ .

Firstly, let us consider the value of  $c$ . Before the move, so in the  $\tau$  form of  $*10*$ , analyzing 2 paths between  $a$  and  $c$  and one between  $b$  and  $c$  we have

$$\begin{aligned} c &= \max\{a + s^1 + p^1, a + p^2 + s^2 + s^1 + p^1, b + p^2 + s^2 + s^1 + p^1\} = \\ &= b + p^2 + s^2 + s^1 + p^1. \end{aligned} \quad (29)$$

After the move, so after changing the form of  $\tau$  into  $\tau'$  of the form  $*01*$ , the  $c$  value changes into  $c'$  as follows:

$$c' = a + s^1 + p^1 \leq b + p^2 + s^2 + s^1 + p^1 = c \quad (30)$$

because  $b \geq a$  (there is an arc from  $a$  to  $b$  in  $G(\delta)$ ).

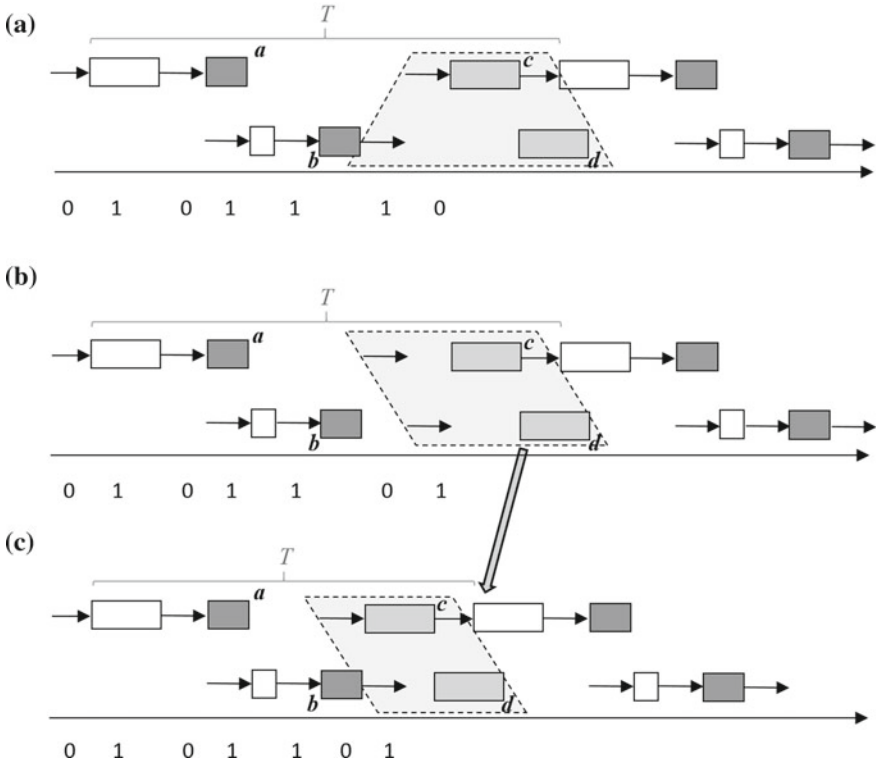


Fig. 3 An example of transforming  $\tau$  from \*10\* (a) to \*01\* (b, c)

Fig. 4 Transforming  $\tau$  from \*10\* to \*01\*—graph model

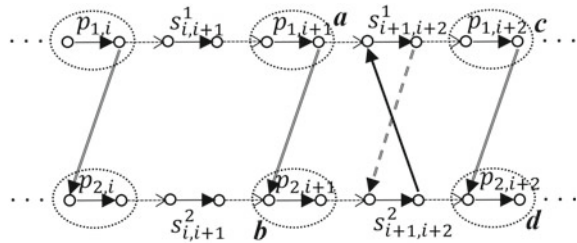
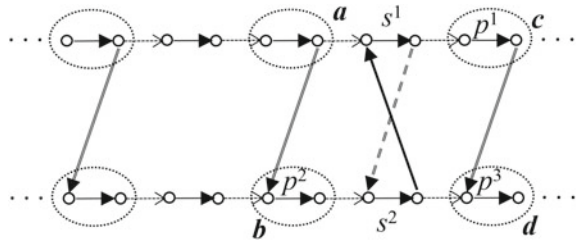


Fig. 5 Transforming  $\tau$  from \*10\* to \*01\*—simplified graph model



For the  $d$  value, before the move, analyzing 5 paths between  $a$  and  $d$  and  $b$  and  $d$  we have

$$d = \max\{a + s^1 + p^1 + p^3, a + p^2 + s^2 + p^3, a + p^2 + s^2 + s^1 + p^1 + p^3, \\ b + p^2 + s^2 + s^1 + p^1 + p^3, b + p^2 + s^2 + p^3\} = b + p^2 + s^2 + s^1 + p^1 + p^3. \quad (31)$$

Analyzing the value of the time moment  $d$  after the move (denoted as  $d'$ ) we have

$$d' = \max\{a + s^1 + p^1 + p^3, a + s^1 + s^2 + p, a + p^2 + s^2 + p^3, b + p^2 + s^2 + p^3\} \leq \\ \leq b + p^2 + s^2 + s^1 + p^1 + p^3 = d. \quad (32)$$

So from Eqs. (30) and (32) we have that  $c \leq c'$  and  $d \leq d'$ , therefore finally we have

$$T(\delta') \leq T(\delta). \quad (33)$$

■

*Remark 2.2* If  $p_{i,j} > 0$ , then the proof of the Theorem 2.1 follow us to the Eq. (28) with strong inequality, that is the minimal cycle time fulfills the inequality

$$T(\delta') < T(\delta). \quad (34)$$

## 5 Case Study

A solution  $\delta = (\pi, \tau)$  can be naturally represented as a pair of permutation  $\pi$  and assignment  $\tau$ . These two object are independent: i.e. there is no need to take under consideration one of the them while changing other. This implies a two-level approach to solve the whole problem. Firstly, change permutation (e.g. by a metaheuristics), secondly—assignment of setups order onto setup team. The second object is dependent of the permutation, because setups are sequence-dependent. However, in some cases they are independent, what shows the following example.

*Example 2.1* Let us consider an instance the considered 2-machine flow shop problem with sequence-dependent disjoint setups of  $i = 3$  jobs with the following parameters of operations duration  $p_{i,j}$ ,  $i = 1, 2, 3$ ,  $j = 1, 2$  and setups duration  $s_{i,k}^j$ ,  $i, k = 1, 2, 3$ ,  $j = 1, 2$ :

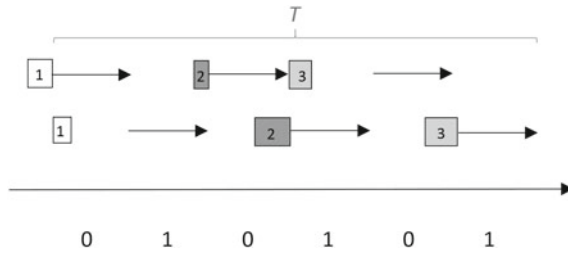


Fig. 6 Gantt chart for the Example—a single MPS

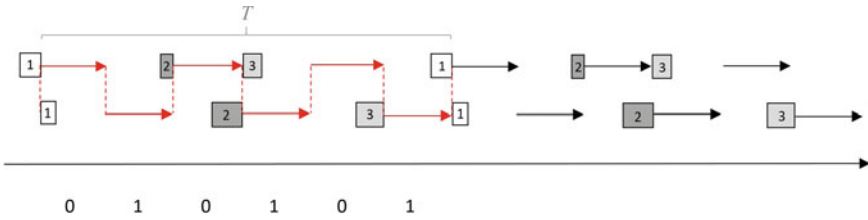


Fig. 7 Cyclic Gantt chart for the Example (2 MPSes)—critical path (red) goes only through setups

$i$	$p_{i,1}$	$p_{i,2}$
1	5	3
2	2	8
3	5	7

$s_{i,k}^1$	1	2	3
1	0	15	15
2	20	0	20
3	17	17	0

$s_{i,k}^2$	1	2	3
1	0	18	18
2	22	0	22
3	21	21	0

We will consider natural (initial) permutation as the jobs order  $\pi = (1, 2, 3)$  and setup team route  $\tau = (010101)$ , as in Fig. 6 (Fig. 7).

The minimal cycle time  $T = 113$  has been computed with the use of Gurobi<sup>1</sup> solver. Both the model (Fig. 8) and the instance data (Fig. 9) have been coded in AMPL<sup>2</sup> modeling language.

The value of the minimal cycle  $T = 113$  and it is the sum of setup times only. We can notice, that this value does not change if we change the permutation  $\pi$ , because all the operations are not critical. We can see it in Fig. 7. Critical path goes only through the setups, keeping operations non-critical. So, for this assignment  $\tau = (010101)$  the permutation does not matter—we will have the same value 113 of the minimal cycle time for any from  $3! = 6$  permutations  $\pi$ .

<sup>1</sup>Mathematical Programming Solver, <http://www.gurobi.com/>.

<sup>2</sup>A Mathematical Programming Language, <https://ampl.com/>.

```

param n > 0 integer;
param m > 0 integer;

param tau {1..2*n} integer;
param sigma {1..2*n} integer;
param theta {1..2*n} integer;

param setup {1..m,1..n,1..n} integer;
param t {1..m,1..n} > 0 integer;

var T >= 0;
var S {1..m,1..n} >= 0;
var P {1..m,1..n} >= 0;

minimize CycleTime: T;

subj to subj01 {j in 2..n}:
    S[1,j] >= P[1,j-1] + setup[1,j-1,j];

subj to subj02 {j in 2..n}:
    S[2,j] >= P[2,j-1] + setup[2,j-1,j];

subj to subj03:
    S[1,1] >= P[1,n] + setup[1,n,1] - T;

subj to subj04:
    S[2,1] >= P[2,n] + setup[2,n,1] - T;

subj to subj05 {j in 1..n}:
    P[1,j] >= S[1,j] + t[1,j];

subj to subj06 {j in 1..n}:
    P[2,j] >= S[2,j] + t[2,j];

subj to subj07 {j in 1..n}:
    S[2,j] >= S[1,j] + t[1,j];

subj to subj08 {j in 1..2*n}:
    P[tau[j],sigma[j]] >= 0;

subj to subj09 {j in 2..2*n}:
    P[tau[j],sigma[j]] >= P[tau[j-1],sigma[j-1]] +
        setup[tau[j-1],sigma[j-1],theta[j-1]];

subj to subj10:
    P[tau[1],sigma[1]] >= P[tau[2*n],sigma[2*n]] +
        setup[tau[2*n],sigma[2*n],sigma[1]] - T;

```

**Fig. 8** AMPL model



**Fig. 9** AMPL data

```

data;

param n := 3;
param m := 2;

param tau : 1 2 3 4 5 6:=
            1 1 2 1 2 1 2;

param sigma : 1 2 3 4 5 6:=
             1 1 1 2 2 3 3;

param theta : 1 2 3 4 5 6:=
             1 2 2 3 3 1 1;

param setup :=
  [1, *, *] 1 1 0 1 2 15 1 3 15
            2 1 20 2 2 0 2 3 20
            3 1 17 3 2 17 3 3 0

  [2, *, *] 1 1 0 1 2 18 1 3 22
            2 1 22 2 2 0 2 3 22
            3 1 21 3 2 21 3 3 0;

param t : 1 2 3:=
         1 5 2 5
         2 3 8 7;

```

## 6 Computational Experiments

Three simple canonical improvement algorithms have been implemented for testing: local search (LS), iterated local search (ILS) and random search (RS) for the examined cyclic two machine flow shop with disjoint sequence-dependent setups. Iterated local search has iteratively repeated the local search procedure starting from random start solutions. Implementations have been done in C++ language. Computational experiments were performed on the Bem<sup>3</sup> cluster with Intel Xeon E5-2670 (2.3 GHz) working under 64-bit CentOS 6.10 (Final) operating system. The matter of the research was to check what is the influence of the assignment (setup team route) onto the value of the goal function. Natural permutation  $\pi$ ,  $\pi(i) = i$ , was fixed for the experiments, so examined procedures change the route (binary sequence  $\tau$ ) of the setup team only. The starting point was chosen as  $\tau = 0101 \dots 0101$ , and it was also a reference solution in comparison. The calculations were performed on test data for the hybrid flow-shop problem proposed in the work of Ruiz and Stützle [11] limited to two machines (only the parameters of first two machines and their setups were taken). In Table 1 particular column means:

---

<sup>3</sup>Calculations have been carried out using resources provided by Wrocław Centre for Networking and Supercomputing (<http://wcss.pl>), grant No. 096.

**Table 1** Results of computer experiments

Problem	$n$	$PRD_{LS}$	$t_{LS}[s]$	$PRD_{ILS}$	$t_{ILS}[s]$	$PRD_{RS}$	$t_{RS}[s]$
DD_SDST001-010	20	-4.34	0.10	-5.91	0.10	-5.58	0.10
DD_SDST011-020	20	-2.69	0.10	-4.27	0.10	-3.82	0.10
DD_SDST021-030	20	-3.14	0.10	-5.14	0.10	-4.62	0.10
DD_SDST031-040	50	-2.46	0.49	-5.58	0.48	-3.86	0.49
DD_SDST041-050	50	-2.97	0.49	-5.80	0.48	-4.11	0.49
DD_SDST051-060	50	-2.93	0.50	-4.93	0.48	-3.99	0.50
DD_SDST061-070	100	-2.66	1.73	-5.51	1.70	-3.69	1.73
DD_SDST071-080	100	-2.59	1.73	-5.11	1.69	-3.85	1.76
DD_SDST081-090	100	-1.82	1.72	-3.83	1.70	-2.85	1.75
DD_SDST091-100	200	-2.76	6.55	-5.37	6.47	-4.14	6.65
DD_SDST101-110	200	-3.09	6.55	-5.46	6.44	-4.11	6.66
DD_SDST111-120	500	-2.77	43.34	-4.48	42.94	-3.79	43.16
Average		-2.85	5.28	-5.11	5.22	-4.03	5.29

- problem—name of the problem instance,
- $n$ —size of the problem instance,
- $PRD_{alg}$ —Percentage Relative Deviation to reference solution given by the formula

$$PRD = \frac{F_{ref} - F_{alg}}{F_{ref}} \cdot 100\% \quad (35)$$

where  $F_{ref}$  is reference goal function value ( $\pi$ —natural permutation,  $\tau = 0101 \dots 0101$ —route of the setup team) and  $F_{alg}$  is the result obtained by the examined local search algorithm operating on binary part of the solution  $\tau$ , and  $alg \in \{LS, ILS, RS\}$ ,

- $t_{alg} [s]$ —computational time of the examined  $alg$  algorithm, where  $alg$  has one of three meanings:  $LS$ —local search algorithm,  $ILS$ —iterated local search algorithm, and  $RS$ —random search algorithm.

Experiments show the possibility of a solution improvement by changing setup team route—up to over 5% of the minimal cycle time, in the short time—except  $n = 500$ , for which the time of algorithm work reaches over 43 s. The conducted experiments show the potential of the second level of optimization: local improvement of the assignment (LS), also together with random starting points (ILS), or random search (RS) alone. As it is visible in the Table 1 the best results have been obtained by iterated local search (ILS) procedure. The natural next step is to implement effective high-level metaheuristics, which optimizes permutational part of solution.

## 7 Remarks and Conclusions

The problem considered in this chapter is quite new and its cyclical version has not been analyzed in the literature so far. In this chapter, we proposed a method of cyclical modeling based on a graph based on a dual representation: permutation and binary sequence. Setting up a problem, allowing only one setup at a given time, complicates the problem quite strongly. A further, natural step is to consider the multi-machine version of the problem ( $m > 2$ ) and more than 1 number of setup teams. There is a great probability that for such a version of the problem—just like for a job shop—the fractional cycle times appear, which is not yet the case discussed here.

## References

1. Bożejko, W.: Solving the flow shop problem by parallel programming. *J. Parallel Distrib. Comput.* **69**(5), 470–481 (2009)
2. Bożejko, W.: Parallel algorithms of discrete optimization in manufacturing. Academic Publishing House EXIT (2018)
3. Bożejko, W., Uchroński, M., Wodecki, M.: Block approach to the cyclic flow shop scheduling. *Comput. Ind. Eng.* **81**, 158–166 (2015)
4. Bożejko, W., Uchroński, M., Wodecki, M.: Parallel metaheuristics for the cyclic flow shop scheduling problem. *Comput. Ind. Eng.* **95**, 156–163 (2016)
5. Corwin, B.D., Esogbue, A.O.: Two machine flow shop scheduling problems with sequence dependent setup times: a dynamic programming approach. *Nav. Res. Logist. Q.* **21**(3), 515–524 (1974)
6. Graham, R.L., Knuth, D.E., Patashnik, O.: *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Reading (1989)
7. Gupta, J.N.: A search algorithm for the generalized flowshop scheduling problem. *Comput. Oper. Res.* **2**(2), 83–90 (1975)
8. Gupta, J.N., Darrow, W.P.: The two-machine sequence dependent flowshop scheduling problem. *Eur. J. Oper. Res.* **24**(3), 439–446 (1986)
9. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. In: *Proceedings of the 1961 16th ACM National Meeting, ACM '61*, pp. 71.201–71.204. ACM, New York (1961)
10. Ruiz, R., Maroto, C.: A comprehensive review and evaluation of permutation flowshop heuristics. *Eur. J. Oper. Res.* **165**(2), 479–494 (2005)
11. Ruiz, R., Stützle, T.: An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *Eur. J. Oper. Res.* **187**(3), 1143–1159 (2008)
12. Taillard, E.: Some efficient heuristic methods for the flow shop sequencing problem. *Eur. J. Oper. Res.* **47**(1), 65–74 (1990)

# Cyclic Scheduling in the Manufacturing Cell



Wojciech Bożejko , Jarosław Pempera , Czesław Smutnicki   
and Mieczysław Wodecki 

**Abstract** The chapter is devoted to scheduling of jobs performed by machines and by an operator in the automated manufacturing cell, which produces parts in large production batches. The purpose of scheduling is to determine a cyclical schedule that minimizes production cycle time. The chapter presents the original model of the problem that enables effective determination of cycle time for any sequence of operations in the cell. What is more, there was an algorithm proposed that determines the sequence and schedule of works minimizing the production cycle time.

## 1 Introduction

Production cells consisting of a sequence of flexible machines are the basis for the construction of modern production systems. Regular surface (usually in the form of a rectangle) that they occupy significantly facilitates the design of a sequence of such cells and designation of communication routes supplying cells with products for processing. Production sockets can be easily adapted to the production of various

---

W. Bożejko · J. Pempera

Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland  
e-mail: [wojciech.bozejko@pwr.edu.pl](mailto:wojciech.bozejko@pwr.edu.pl)

J. Pempera

e-mail: [jaroslaw.pempera@pwr.edu.pl](mailto:jaroslaw.pempera@pwr.edu.pl)

C. Smutnicki

Department of Computer Engineering, Faculty of Electronics, Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland  
e-mail: [czeslaw.smutnicki@pwr.edu.pl](mailto:czeslaw.smutnicki@pwr.edu.pl)

M. Wodecki (✉)

Department of Telecommunications and Teleinformatics, Faculty of Electronics, Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland  
e-mail: [mieczyslaw.wodecki@pwr.edu.pl](mailto:mieczyslaw.wodecki@pwr.edu.pl)

© Springer Nature Switzerland AG 2020

W. Bożejko and G. Bocewicz (eds.), *Modelling and Performance Analysis of Cyclic Systems*, Studies in Systems, Decision and Control 241,  
[https://doi.org/10.1007/978-3-030-27652-2\\_3](https://doi.org/10.1007/978-3-030-27652-2_3)

products. This is done by changing of the configuration of flexible machines, heads and execution tools or, as a last resort, replacing the entire machine.

Modern automated production cells usually consist of electronically controlled flexible machines, which due to electronic control can be quickly adapted to the production of various parts. One of the benefits of connecting machines to production cells is reduction of the activities related to material handling. Inside the cell, moving parts between the machines is usually done by one (or several) robots, while the parts between the cells are transported by automatic guided vehicles (AGV).

The element that coordinates the performance of technological activities in a fully automated cell is the robot. Nevertheless, in many areas of production robots are often replaced by an operator. In particular, this applies to cells in which production is performed sporadically and requires constant supervision of the employee, or requires the operator to precisely place a large product in the machine handles, or the operator performs certain technological steps in a manual manner (e.g. grinding, precision welding, etc.)

Classical models for scheduling of production tasks seem to be unsuitable for use in scheduling production cells, as they generally do not take into account the most important feature of flexible production cells, namely, the interaction between the transport system (robot, operator) and machines. In addition, there are relatively few papers in the literature devoted to the cyclic production strategy, which is the basis for effective control of the automated production cell.

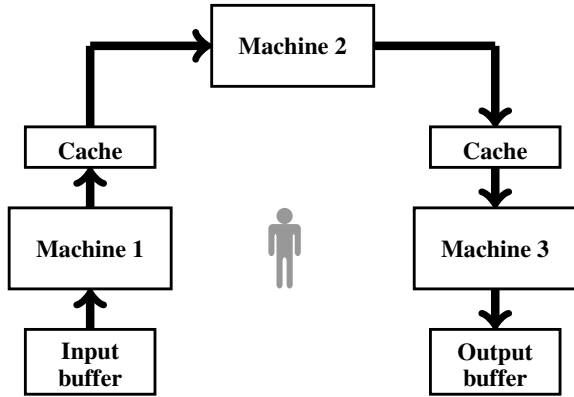
Various variants of robotic production cells are proposed in the literature. These variants differ in the number and type of robots, the number of different parts machined in one production takt, time requirements, etc. In the work [5] a robotic production cell operated by two means of transport is considered. The algorithms for controlling the production cell supported by many robots with double heads are proposed in the work [7]. Works [1, 6] are devoted to the scheduling of production cells in which many different products are made at each production cycle at the same time.

Cyclic scheduling with time windows of production cells has been examined in the paper [11]. Theoretical results concerning production cells were actually applied in the industry. In the paper [9] there was proposed an algorithm to support scheduling in Electroplating Facilities, whereas in [2] there was proposed an exact algorithm to find minimal number of workers to service the cells in a factory producing car seats.

## 2 Problem Description

The production cell consists of the input buffer,  $m$  machines from the set  $M = \{1, 2, \dots, m\}$  and the output buffer. Between the machines there are storage places (cache) on which one can store at most one part, see Fig. 1. The processed part is taken from the input buffer, then is processed on each machine in order according to their numbering; and eventually the part is placed in the output buffer. The production cell is operated by one operator. The cell operator performs all the moves of the parts

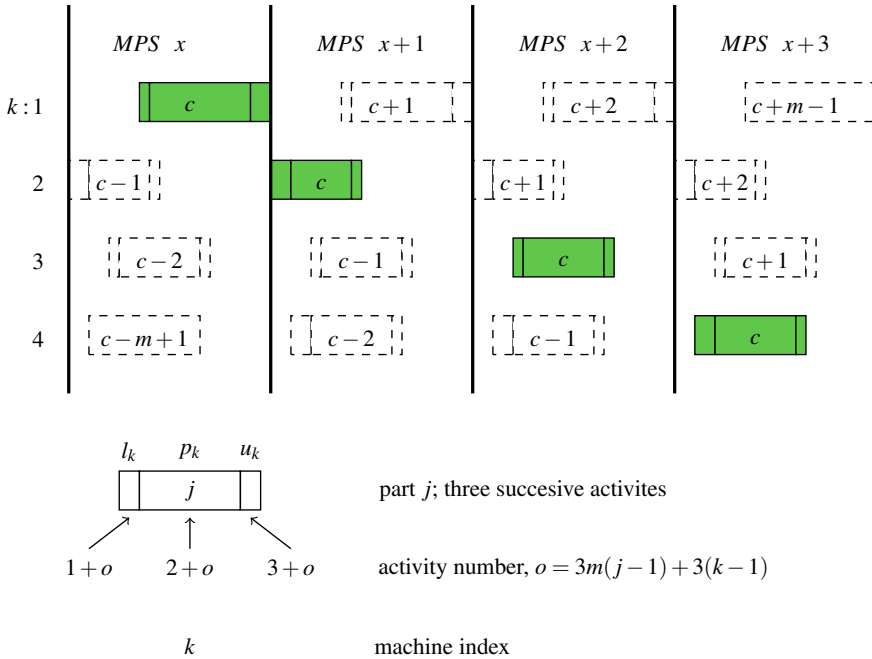
**Fig. 1** Structure of manufacturing cell



in the cell, the works of mounting parts on the machines and unloading them from the machines. In the time of loading and/or unloading the element on the machine cannot perform technological operations. After unloading the machine, the part is placed in the storage box or loaded on the next machine if it is empty.

In the production system there must be  $n$  identical production tasks performed. Each task requires processing on each machine. The time of processing of parts on the  $k$  machine is  $p_k > 0$ . The time  $l_k > 0$  of fixing the part on the machine  $k$  is given and the time of its removal  $u_k > 0$  from the machine. Fastening parts on the machine is possible only when it is empty, so immediately after installation there can be processing of parts on the machine performed. The machine is blocked by a part (it cannot process the next part) until the part is removed. During the assembly of parts on the machine, the time of all activities requiring the presence of the operator before the actual treatment (not requiring operator's supervision) is included. In addition to fastening the parts, there can be adjustments to machine settings, machine start control, etc. performed. On the other hand, during the unloading period, apart from the time needed to remove parts from the machine, the total time of activities such as quality control, removal of minor defects or putting away the storage field is included.

For each machine pair  $i, k \in \{1, \dots, m\}$  the time consumed for moves of the operator is given  $D_{ik} \geq 0$ . The transport time from the input buffer to the first machine and the transport time from the last machine to the output buffer are included in the time of loading the first and unloading the last machine respectively. It has been assumed that the time in which the operator moves with the product and without the product (empty) is identical. In the input and output buffer, parts are stored in large quantities (we assume that their capacity is infinitely large). In other parts of the cell (machines, storage field) at the moment there may be only one part. There are three activities performed on each machine: (i) placing the product on the machine, (ii) processing, (iii) removing of the product from the machine and putting it in the storage field. Thus, for each task, there should be  $3m$  activities performed on the machines.



**Fig. 2** Policy of realising activities in cycles  $x, x + 1, \dots, x + 3$  for  $m = 4$

From the manufacturing policy we conclude, that the production cell achieves the greatest efficiency when in each production takt, on each machine a different part is made (no machines with outage), i.e. on  $m$ th machine there is part 1 made, on the machine  $m - 1$  part 2, etc. finally on the machine 1 part  $m$ . Despite the parts are identical, we assume hereinafter, that they are indexed by successive integers in the order of processing. Let us consider the sequence of takts  $x = c, c + 1, c + 2, \dots$  for certain  $c$ , see Fig. 2. In the takt  $x = c$  machines 1, 2,  $\dots, m$  perform parts denoted by indexes  $c, c - 1, \dots, c - m + 1$ , respectively. In the next takt there are parts with numbers enlarged by one.

Performing of one part (denoted in Fig. 2 by  $c$ ) needs  $m$  successive takts, thence without losing generality we can analyse only cycles  $x = c, c + 1, \dots, c + m - 1$ , also for  $c = 1$ . The execution of the production takt described above requires the execution of  $m - 1$  initial takts, in which 1 to  $m - 1$  parts are processed, respectively. The schedule for performing activities in these takts will be omitted. The operator may perform only one activity at a time, therefore it is necessary to determine the order of performing these activities by the operator.

The cyclic production strategy assumes repeated execution of identical operations in the same order. The set of works performed in one takt will be called the cyclic core. In addition, for each pair of corresponding actions in two consecutive cycles,

the difference in the moments of their start must be identical. This period is called the cycle time and we will be denoted by the symbol  $T$ .

It is easy to note that in every cycle there is performed the last technological operation of a product, i.e. the production cell leaves one product in each cycle. Therefore, increasing the productivity of the line can be achieved by reducing the time of the production cycle.

### 3 Mathematical Model

Manufacturing of  $n$  part in a production cell consisting of  $m$  machines require  $n \cdot 3m$  activities. Let us assume that activities with numbers  $o_j + 1, o_j + 2, \dots, o_j + 3m$ , where  $o_j = (j - 1) \cdot 3m$  correspond to processing in the cell of  $j$ th part. Activity

$o_{j,k}^1 = o_j + 3(k - 1) + 1$  corresponds to fixing of  $j$ th part on machine  $k$ ,  
 $k = 1, 2, \dots, m$ ,

$o_{j,k}^2 = o_j + 3(k - 1) + 2$  corresponds to processing of  $j$ th part on machine  $k$ ,  
 $k = 1, 2, \dots, m$ ,

$o_{j,k}^3 = o_j + 3(k - 1) + 3$  corresponds to removing of  $j$ th part from machine  $k$ ,  
 $k = 1, 2, \dots, m$ .

All operations require the machine to be involved, so on the  $k$  machine the operations are performed in the following order

$$\pi_k = (o_{1,k}^1, o_{1,k}^2, o_{1,k}^3, o_{2,k}^1, o_{2,k}^2, o_{2,k}^3, \dots, o_{n,k}^1, o_{n,k}^2, o_{n,k}^3), \quad k = 1, 2, \dots, m. \quad (1)$$

The cyclic core will be called a set of all activities performed in the 1st production takt in which the processing of parts is being performed on all machines. This set consists of the following operations:

$$\mathcal{C} = \bigcup_{k=1}^m \left\{ o_{(m-k+1),k}^1, o_{(m-k+1),k}^2, o_{(m-k+1),k}^3 \right\}. \quad (2)$$

It is easy to see that in  $x$ th cycle with respect to 1th cycle there are operations performed with numbers increased by  $(x - 1) \cdot 3m$ .

The order of executing core operations  $\mathcal{C}$  on machines can be described as follows:

$$\alpha = (\alpha_1, \dots, \alpha_m), \quad (3)$$

where

$$\alpha_k = (\alpha_k(1), \alpha_k(2), \alpha_k(3)), \quad \alpha_k(i) = o_{(m-k+1),k}^i, \quad k = 1, 2, \dots, m. \quad (4)$$

Notice,  $\alpha$  is fixed, is not a decision variable.



Let  $\mathcal{C}^o \subset \mathcal{C}$  be a subset of the cyclic core operations that require operators presence. The set  $\mathcal{C}^o$  is defined as follows

$$\mathcal{C}^o = \bigcup_{k=1}^m \left\{ o_{(m-k+1),k}^1, o_{(m-k+1),k}^3 \right\}. \quad (5)$$

Next, let us denote by  $\tau = (\tau(1), \tau(2), \dots, \tau(2 \cdot m))$ ,  $\tau(i) \in \mathcal{C}^o$  the order in which the cell operator performs cyclic core operations.

Let  $S_i^x (C_i^x)$  be the earliest starting (completion) moment of the  $i$ th operation execution in  $x$ th cycle. The schedule for performing operations in the production cell, described by the starting  $S_i^x$  and the completion  $C_i^x$  time moments, the operation must meet the following constraints:

$$S_{\alpha_k(i)}^x \geq C_{\alpha_k(i-1)}^x, \quad i = 2, 3, \dots, m, \quad k = 1, 2, \dots, m, \quad x = 1, 2, \dots, \quad (6)$$

$$S_{\alpha_k(1)}^{x+1} \geq C_{\alpha_k(1)-1}^x, \quad k = 2, 3, \dots, m, \quad x = 1, 2, \dots, \quad (7)$$

$$S_{\tau(i)}^x \geq C_{\tau(i-1)}^x + D_{\tau(i-1), \tau(i)}, \quad i = 2, 3, \dots, 2m, \quad x = 1, 2, \dots, \quad (8)$$

$$S_{\alpha_k(1)}^{x+1} \geq C_{\alpha_k(3)}^x, \quad k = 1, 2, \dots, m, \quad x = 1, 2, \dots, \quad (9)$$

$$S_{\tau(1)}^{x+1} \geq C_{\tau(2m)}^x + D_{\tau(m), \tau(1)}, \quad x = 1, 2, \dots, \quad (10)$$

Inequality (6) means that in each cycle, for each part being processed, the starting moment of the machine cannot be earlier than the completion time of loading by operator and the moment of start of unloading cannot be earlier than the moment of finishing the processing. It is easy to notice that processing of a given part on two consecutive machines takes place in two consecutive cycles. Thus, loading parts on the  $k$ th machine in the  $x + 1$  cycle (action  $\alpha_k(1)$ ) can only start after the completion of the previous operation ( $\alpha_k(1) - 1$ ) i.e. unloading parts on the previous machine ( $k - 1$ ) which is carried out in the previous ( $x$ )th cycle. These dependencies are modeled by the constraint (7). Inequality (8) refers to actions performed by the operator and means that  $i$ th action in the order  $\tau$  can start after the end of the previous activity performed by the operator plus transport time.

The remaining inequalities (9), (10) model the sequence relations between operations performed in two consecutive cycles. Inequality (9) means that the first action on  $k$ th machine (unloading of parts) can only start after the last operation on the machine has been completed in the previous cycle (unloading the machine). Because of the lack of possibility to suspend the activities, the moment of their completing is:

$$C_i^x = S_i^x + l_i, \quad i = 1, 3, \dots, 3m - 2, \quad x = 1, 2, \dots, \quad (11)$$

$$C_i^x = S_i^x + p_i, \quad i = 2, 4, \dots, 3m - 1, \quad x = 1, 2, \dots, \quad (12)$$

$$C_i^x = S_i^x + u_i, \quad i = 3, 6, \dots, 3m, \quad x = 1, 2, \dots \quad (13)$$

In addition, cyclic scheduling requires that

$$S_i^{x+1} = S_i^x + T, \quad i = 1, \dots, 3m, \quad x = 1, 2, \dots \quad (14)$$

Ultimately, we want to find the order  $\tau^*$  of performing the action by the operator such that for pair  $(\alpha, \tau^*)$  there is a schedule that meets the inequality (6)–(14) with the  $T$  cycle time the smallest possible. The problem can be decomposed into two subproblems: the problem of the lower level—the determination of the smallest value of  $T$  for the order  $\tau$ , the problem of the upper level—finding the order of  $\tau^*$  with the smallest cycle time.

### 3.1 Example

Consider a production cell consisting of  $m = 4$  machines. Each task consists of  $m = 4$  technological operations and a total of  $3m = 12$  operations performed in the cell. Activity times are summarized in Table 1. It was assumed that transport times between stands are to be omitted, i.e. equal to zero.

The cyclic core is in the form:

$$\alpha = ((37, 38, 39), (28, 29, 30), (19, 20, 21), (10, 11, 12)), \quad (15)$$

where operations 37, 38, 39 correspond to processing of the 4th task on the first machine, operations 28, 29, 30 correspond to processing the 3rd task on the second machine, operations 19, 20, 21 correspond to machining the 2nd task on the third machine, while operations 10, 11, 12 correspond to the processing of the 1st task on the fourth machine. The cell operator performs operations from the core in the order  $\tau = (37, 28, 39, 19, 30, 10, 21, 12)$ .

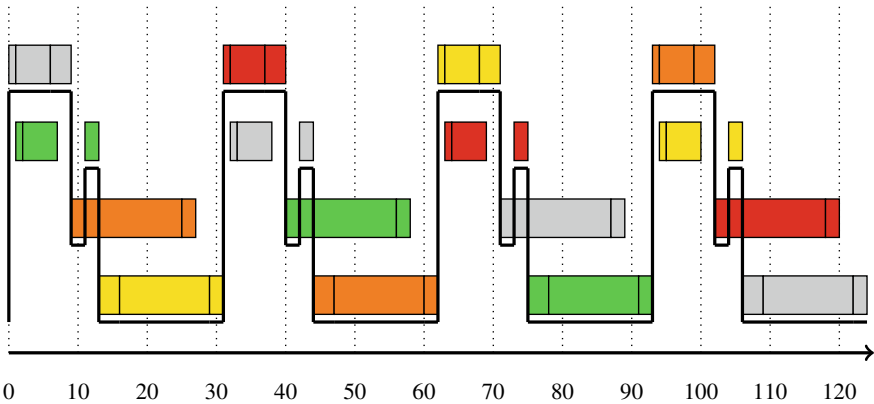
The schedule for performing the cyclic core operation is presented in Table 2. Figure 3 illustrates the schedule for completing 4 cycles with the cycle time  $T = 31$ .

**Table 1** Times of activities performed in the production cell

Parameters	Machines			
	1	2	3	4
$l_i$	1	1	2	2
$p_i$	5	5	14	13
$u_i$	3	2	2	2

**Table 2** A cyclical schedule for performing operations in a production cell

Activity	$S_1^1$	$C_1^1$	$S_2^1$	$C_2^1$	$S_3^1$	$C_3^1$	$S_4^1$	$C_4^1$
Loading	0	1	1	2	9	11	13	16
Processing	1	6	2	7	11	25	16	29
Uploading	6	9	11	13	25	27	29	31



**Fig. 3** Cyclic schedule for the data from the example

## 4 Solution Method

Designating a cyclic schedule for the problems of scheduling production tasks is one of the most difficult challenges for scientists. In these problems, in addition to the order requirements (characteristic for classical problems), there is a (cycle time)—the interval between activities belonging to two subsequent cycles required. While constructing a schedule that meets the order requirements is relatively fast (linear to the number of operations performed), the minimum value of  $T$  requires finding (for example, by binary search method) such a minimum  $T$  for which a schedule meeting the order requirements can be constructed. In such a case, we get an algorithm over polynomial determining cycle time for a given sequence of operations in a cycle. Another way to determine the cycle time is to apply optimization packages that use methods to propagate constraints. The state of today’s research on cyclical problems [8] indicates that it is possible to set cyclic schedules only for small problems with these methods. For larger problems, it is necessary to develop heuristic algorithms [4]. In recent years, a new method for determining the minimum cycle time with polynomial complexity has been developed. A description of this method can be found in the works ([3, 10]). This method can also be used when determining the cycle time in the production cell. It is enough to build a graph that corresponds to the order relationships in  $m$  production cycles.

## 5 Graph Model

Before proceeding with the description of the graph model, we will convert the inequalities (6), (7) replacing the values of  $C_i^x$  with expressions (11)–(13). After transformation, we get

$$S_{\alpha_k(i)}^x - (S_{\alpha_k(i-1)}^x - p'_{\alpha_k(i-1)}) \geq 0, \quad (16)$$

$$i = 2, 3, \dots, m, \quad k = 1, 2, \dots, m, \quad x = 1, 2, \dots,$$

$$S_{\alpha_k(1)}^{x+1} - (S_{\alpha_k(1)-1}^x - p'_{\alpha_k(1)-1}) \geq 0, \quad k = 2, 3, \dots, m, \quad x = 1, 2, \dots, \quad (17)$$

$$S_{\tau(i)}^x - (S_{\tau(i-1)}^x - p'_{\tau(i-1)}) \geq D_{\tau(i-1), \tau(i)}, \quad i = 2, 3, \dots, 2m, \quad x = 1, 2, \dots, \quad (18)$$

$$S_{\alpha_k(1)}^{x+1} - (S_{\alpha_k(3)}^x - p'_{\alpha_k(3)}) \geq 0, \quad k = 1, 2, \dots, m, \quad x = 1, 2, \dots, \quad (19)$$

$$S_{\tau(1)}^{x+1} - (S_{\tau(2m)}^x - p'_{\tau(2m)}) \geq D_{\tau(m), \tau(1)}, \quad x = 1, 2, \dots, \quad (20)$$

where

$$p'_i = l_i, \quad i = 1, 3, \dots, 3m - 2, \quad (21)$$

$$p'_i = p_i, \quad i = 2, 4, \dots, 3m - 1, \quad (22)$$

$$p'_i = u_i, \quad i = 3, 6, \dots, 3m. \quad (23)$$

For a given number of production cycles  $Y$ , the inequality system (16)–(20) can be solved by constructing of a directed graph  $G(\alpha, \tau) = (V, E)$  with a set of nodes  $V$  and a set of weighted arcs  $E$ . The set of nodes consists of  $3mY$  nodes in the form of  $i^x$ ,  $i = 1, 2, \dots, m$ ,  $x = 1, 2, \dots, Y$ . The  $i^x$  node corresponds to the  $i$  operation performed in the  $x$ th cycle and is weighted by  $p'_i$ . The set of arcs  $E$  consists of 5 subsets:

$$A(\alpha) = \bigcup_{x=1}^Y \bigcup_{k=1}^m \{(\alpha_k^x(1), \alpha_k^x(2)), (\alpha_k^x(2), \alpha_k^x(3))\} \quad (24)$$

$$F^* = \bigcup_{x=2}^Y \bigcup_{k=2}^m \{(\alpha_k^x(1) - 1, \alpha_k^x(1))\} \quad (25)$$

arcs from th sets  $A(\alpha)$  and  $F^*$  are loaded with a weight equal to zero.

$$W(\tau) = \bigcup_{x=1}^Y \bigcup_{i=2}^{2m} \{(\tau^x(i-1), \tau^x(i))\} \quad (26)$$

arc  $(\tau^x(i - 1), \tau^x(i)) \in W(\tau)$  are loaded with a weight  $D_{\tau(i-1),\tau(i)}$ .

$$A^*(\alpha) = \bigcup_{x=2}^Y \bigcup_{k=1}^m \{(\alpha_k^{x-1}(3), \alpha_k^x(1))\} \tag{27}$$

arcs from the set  $A^*(\alpha)$  are loaded with a weight equal to zero,

$$W^*(\tau) = \bigcup_{x=2}^Y \{(\tau^{x-1}(2m), \tau^x(1))\} \tag{28}$$

arc  $((\tau^{x-1}(2m), \tau^x(1))) \in W^*(\tau)$  is loaded with a weight  $D_{\tau(2m),\tau(1)}$ .

It is easy to see that sets  $A(\alpha)$  and  $W(\tau)$  model the sequence relationships inside production cycles, while the remaining sets (sets marked with \*) model the sequence relations between cycles.

Figure 4 shows the  $G(\alpha, \tau)$  graph constructed for the operator's order  $\tau = (37, 28, 19, 10, 39, 30, 21, 12)$ . This order describes one of the simplest strategies for operating a production cell in which the operator first places the parts on the machines in the order of their numbering, returns to the first machine and unloads all the machines in the same order.

The  $G(\alpha, \tau)$  graph was constructed for  $m + 1 = 5$  cycles. The Fig. 4 showing 4 full cycles and a fragment consisting of nodes corresponding to the first operations performed on machines in the fifth cycle. The nodes corresponding to the loading

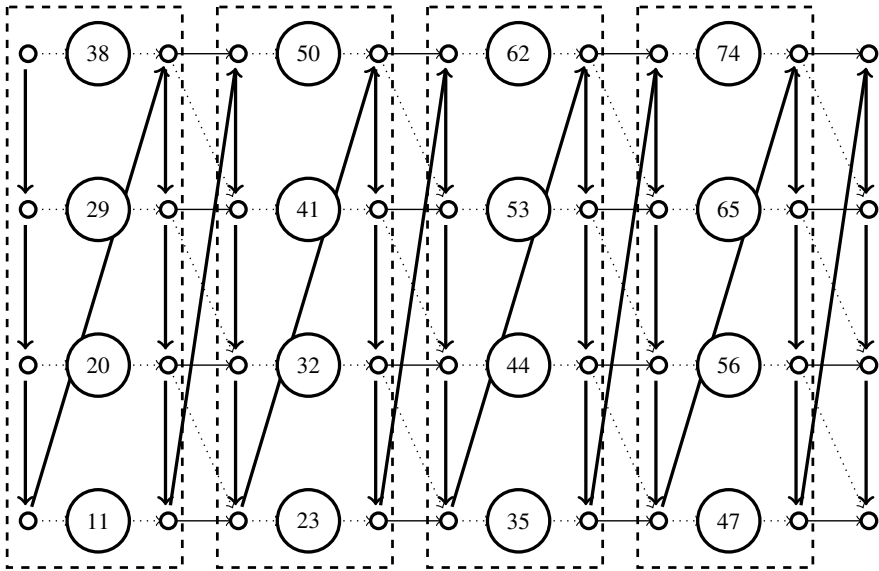


Fig. 4 Graph  $G(\alpha, \tau)$

and unloading operations were drawn with small circles, whereas the nodes corresponding to the operations performed exclusively by the machine are marked with large circles with the numbers of operations in the middle. Fragments of the graph corresponding to the next cycles were framed with a dashed line.

Different types of arcs appearing in the graph were differentiated by the type of line. Arcs representing technological relations from the sets  $A(\alpha)$  and  $F^*$  were marked with a dotted line, arcs from the set  $A^*(\alpha)$  with a thin solid line, whereas arcs depending on the operator's order of performing actions (sets  $W(\tau)$  and  $W^*(\tau)$ ) are marked with a solid line.

Using the commonly known relationship between the longest paths in the graph that model the sequence relations and the earliest moments of starting operations, the following theorem can be formulated.

**Theorem 1** *The order of operations execution by the operator  $\tau$  is allowed if the  $G(\alpha, \tau)$  graph is an acyclic graph.*

For proof, it is enough to note that in the  $G(\alpha, \tau)$  graph all nodes are loaded with positive weights and the arcs have a load of not less than zero. Thus, any cycle in such a graph has a positive length, which implies no solution of the inequality system (6)–(10).

Let us denote by  $L_{s^x, t^y}$ ,  $x < y$  the length of the longest path from node  $s^x$  to node  $t^y$ .

**Theorem 2** *For a permissible sequence of actions performed by the operator  $\tau$  the minimal cycle time is equal to*

$$T(\alpha, \tau) = \max_{2 \leq y \leq m+1} \max_{1 \leq k \leq m} \left( \frac{L_{(\alpha_k(1))^1, (\alpha_k(1))^y}}{y - 1} \right). \quad (29)$$

The proof of the theorem is analogous to the proof of the theorem found in the work [10].

**Theorem 3** *For a given order of  $\tau$  the exact value of the cycle time can be determined in time  $O(m^3)$ .*

Determining the value of  $T(\alpha, \tau)$  from the expression (29) requires the construction of  $G(\alpha, \tau)$  graph consisting of  $m + 1$  cycles. Each cycle has  $3 \cdot m$  nodes, so the graph consists of  $3 \cdot m^2$  nodes. Further, the graph  $G(\alpha, \tau)$  is rare (with the number of arcs leaving the node not bigger than 4) therefore, we can use the well-known algorithm for determining the longest paths in DAG with the complexity  $O(3 \cdot m^2)$  to determine the length of the longest paths. Finally, the algorithm should be executed  $m$ -times for the source nodes  $\alpha_k(1)$ ,  $k = 1, 2, \dots, m$ .

### 5.1 Determining of Optimal Order

In real production cells, the operator (robot) usually handles a small number of machines at the same time. Thus, the number of all possible orders, although it is  $2m!$ , is relatively small for the computing capabilities of modern PCs. Additionally, having a time-efficient method of determining the cycle time with the computational complexity  $O(m^3)$  we decided to use the Brute-force algorithm. In order to speed up the performance of the algorithm, generating solutions not accepted in an obvious way i.e. those in which the unloading of the machine was performed before loading the machine, was prevented. The running time of the algorithm for the number of machines not more than 5 did not exceed a few seconds.

One of the optimal solutions determined for the data from the example is the order  $\tau = (19, 10, 28, 37, 30, 39, 21, 11)$ . The cycle time for this order is 20 and it is 2 less than the cycle time determined for the operator's simple strategy of acting discussed in the previous section. The  $G(\alpha, \tau)$  graph for this order is shown in Fig. 5.

The schedule for performing production activities for both the simple strategy (the upper part of the chart) and the optimal strategy (bottom part of the chart) has been presented in the Gantt chart (see Fig. 6). A critical path is marked with a thick line, while the numbers indicate the numbers of operations performed by the machine. The analysis of both charts shows that in both cases the production cells do not work at full capacity, i.e. in both cases we observe machine downtimes.

In the optimal strategy, the smallest downtime is observed on the machine 3, it amounts to 2 and is related to the unloading of the fourth machine. In a simple strategy, this time is by 2 bigger, i.e. the operator, apart from unloading the fourth

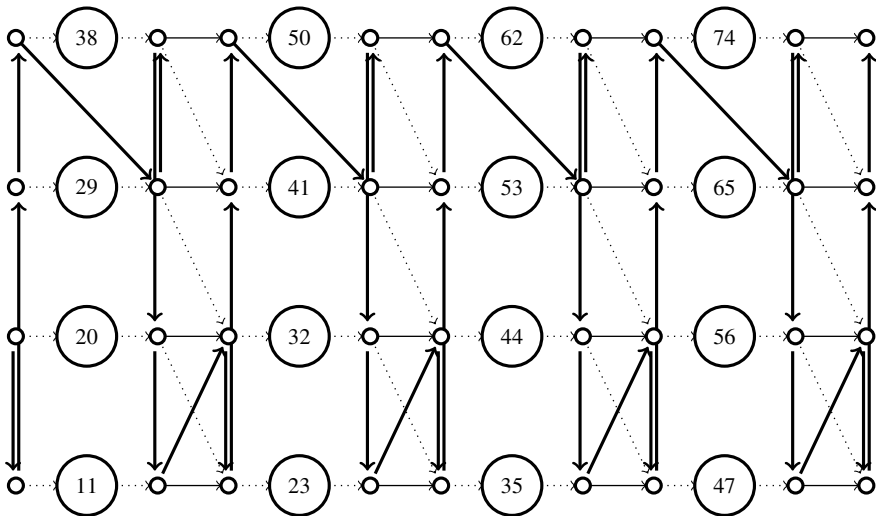
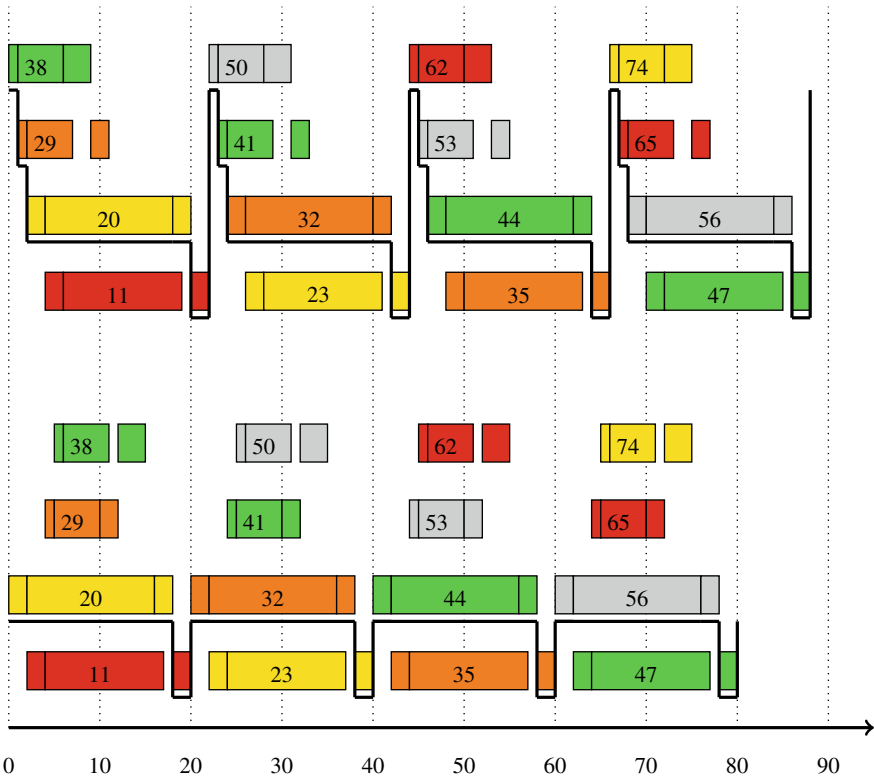


Fig. 5 Graph  $G(\alpha, \tau)$  for optimal  $\tau$



**Fig. 6** Cyclic schedule for simple and optimal operator order

machine, must still load machines 1 and 2. From the presented charts, we can read other important information regarding both strategies. The simple calculations show that the operator performs activities that together last 15, which is significantly lower value than the theoretical capacity of the cell which is 18 (see the sum of activities on the machine 3), so it may seem that the designation of the operator’s schedule of operations is trivial. Moreover, performing activities in accordance with a simple strategy, apart from generating a schedule with a longer cycle time, is additionally burdened with a high risk of its elongation because as many as five operations performed by the operator are critical operations. In the case of an optimal strategy, these are only three activities.

## 6 Summary

The chapter presents a new method, based on elements of graph theory, modeling of a production cell operated by one operator. Based on the graph model, a number of properties were formulated, which were used to construct an algorithm that sets



the minimum cycle time for a given sequence of actions performed by an operator with the complexity of  $O(m^3)$ . In the case of production cells consisting of several machines, the optimal order of operations by the operator can be determined in a few seconds on a PC computer by Brute-force algorithms.

The algorithm proposed in the work can be used not only to diagnose reasons of poor performance but also its improvement in operating production cells, while designing production cells in manufacturing process of new products, testing new production diagrams in the cell, etc.

## References

1. Batur, G.D., Karasan, O.E., Akturk, M.S.: Multiple part-type scheduling in flexible robotic cells. *Int. J. Prod. Econ.* **135**(2), 726–740 (2012)
2. Bożejko, W., Pempera, J., Wodecki, M.: Minimization of the number of employees in manufacturing cells. In: Graña, M., López-Guede, J.M., Etxaniz, O., Herrero, Á., Sáez, J.A., Quintián, H., Corchado, E. (eds.) *International Joint Conference SOCO'18-CISIS'18-ICEUTE'18*, pp. 241–248. Springer (2019)
3. Bożejko, W., Pempera, J., Wodecki, M.: Minimal cycle time determination and golf neighborhood generation for the cyclic flexible job shop problem. *Bull. Pol. Acad. Sci.: Tech. Sci.* **66**(3), 333–344 (2018)
4. Brucker, P., Kampmeyer, T.: Cyclic job shop scheduling problems with blocking. *Ann. Oper. Res.* **159**, 161–181 (2008)
5. Chtourou, S., Manier, M.A., Loukil, T.: A hybrid algorithm for the cyclic hoist scheduling problem with two transportation resources. *Comput. Ind. Eng.* **65**(3), 426–437 (2013)
6. Elmi, A., Topaloglu, S.: Multi-degree cyclic flow shop robotic cell scheduling problem: ant colony optimization. *Comput. Oper. Res.* **73**(C), 67–83 (2016)
7. Galante, G., Passananti, G.: Minimizing the cycle time in serial manufacturing systems with multiple dual-gripper robots. *Int. J. Prod. Res.* **44**(4), 639–652 (2006)
8. Kampmeyer, T.: Cyclic scheduling problems. Ph.D. thesis, University Osnabriick (2006)
9. Manier, M.A., Lamrous, S.: An evolutionary approach for the design and scheduling of electroplating facilities. *J. Math. Model. Algorithms* **7**(2), 197–215 (2008)
10. Pempera, J., Smutnicki, C.: Open shop cyclic scheduling. *Eur. J. Oper. Res.* **269**(2), 773–781 (2018)
11. Yan, P., Chu, C., Yang, N., Che, A.: A branch and bound algorithm for optimal cyclic scheduling in a robotic cell with processing time windows. *Int. J. Prod. Res.* **48**(21), 6461–6480 (2010)

# On Estimating LON-Based Measures in Cyclic Assignment Problem in Non-permutational Flow Shop Scheduling Problem



Andrzej Gnatowski  and Teodor Niżyński 

**Abstract** In recent years, Fitness Landscape Analysis (FLA) has provided a variety of new methods to analyze problem instances, allowing for a better understanding of the challenges that operations research is facing. Many from the most promising FLA methods are based on Local Optima Networks (LON), a compact representation of a search space from the perspective of a optimization algorithms. In order to obtain a representative LON, a solution space sampling procedure must be utilized. However, there is little known about the proper sampling methods—as well as the minimal amount of computational effort required to *sufficiently* sample the space. In this chapter, we investigate the impact of the number of samples taken, on the obtained LON metrics for Cyclic Assignment Problem in non-permutational Flow Shop Scheduling Problem. The sampling process is performed in incremental steps, until the entire solution space is analyzed. After each step, LON measures are calculated. The results suggest a strong relation between the measure values and sampling effort.

## 1 Introduction

The presence in the industry of the manufacturing systems with parallel machines [4, 5, 24] can be explained both by the need to increase the production capacity and the growing demand for a diversified final product. With the rapid development of a computing technology, it has become possible to effectively manage flexible production systems in which tasks can be performed in many alternative ways. Due to the unpredictability and relatively large diversification of a production profile, it is

---

A. Gnatowski (✉) · T. Niżyński  
Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław  
University of Science and Technology, 27 Wybrzeże Wyspiańskiego St.,  
50-372 Wrocław, Poland  
e-mail: [andrzej.gnatowski@pwr.edu.pl](mailto:andrzej.gnatowski@pwr.edu.pl)

T. Niżyński  
e-mail: [teodor.nizynski@pwr.edu.pl](mailto:teodor.nizynski@pwr.edu.pl)

© Springer Nature Switzerland AG 2020  
W. Bożejko and G. Bocewicz (eds.), *Modelling and Performance Analysis  
of Cyclic Systems*, Studies in Systems, Decision and Control 241,  
[https://doi.org/10.1007/978-3-030-27652-2\\_4](https://doi.org/10.1007/978-3-030-27652-2_4)

often necessary to use machines with different parameters, making alternative technological routes possible. The task of optimizing such complex processes requires the use of sophisticated models and dedicated algorithms. Therefore, in order to obtain high-quality solutions, it is necessary to identify not only problem-specific properties, but also an instance-level ones.

In the paper we investigate one of well-known methods of instance analysis—Local Optima Networks (LON) analysis, applied to cyclic assignment problem in non-permutational Flow Shop Problem. LON analysis is a part of a larger methodology, Fitness Landscape Analysis (FLA). The main obstacle preventing the practical application of the said method is the large amount of computational effort necessary to obtain a representative image of the solution space. The goal of this research is to examine the impact of solution space sampling on the values of LON measures.

## 1.1 Production Systems

The concept of cellular manufacturing combines two ways of organizing a production system: the one focusing on the most efficient production of a homogeneous product (*product layouts*, e.g. production lines) and the one that puts emphasis on the flexible use of different functionalities of individual production assets (*functional layouts*, as in a job shop or Flow Shop Problem). Production cells are usually clusters of machines capable of performing various operations. They produce a family of products, sharing similar technological demands. In the production cells defined in such a way, robots are often used to transport the manufactured elements between the machines. More information on cell manufacturing, including the use of robots, can be found in [8].

Cyclic flow shop robotic cells are among the most commonly studied robotic cell optimization problems. Such cell consists of  $m$  machines and 1, 2 or  $r$  robots used to transport the manufactured elements (jobs) between the machines, where operations are performed. The production is repeated an infinite amount of times in a cyclic manner. Since the earliest works, cyclic robotic cell optimization problems have been divided into two groups [25]: those with one and those with multiple different product types (jobs) to be manufactured. The one-job variant is the most researched one. An overview of such problems can be found in [6, 10] and [7]. Depending on the additional constraints, some of the problems are polynomially solvable. Optimization in robotic cells with multiple different jobs is often a harder task. Literature review on this topic can be found e.g. in [9], where single-robot robotic cell was studied. Both exact (Mixed Integer Programming formulation) and approximate algorithms (parallel tabu search and genetic algorithm) for the problem were proposed. The impact of a definition of cyclicity on production efficiency was also examined.

In this paper, we consider a manufacturing system with one operator (machine) in each production cell. The system is a hybrid of a cyclic assignment problem [2] and non-permutational Flow Shop Problem. An example of optimization algorithm for the problem was shown in [1], where a two-level approach was taken. The operation to machine assignments were obtained by both approximate and exact algorithms, whereas the schedules were optimized by a tabu search algorithm.

## 1.2 Fitness Landscape Analysis

Fitness Landscape (FL) is, for a given instance of a problem, a structured representation of a solving algorithm search space. By analyzing FL it is possible to grasp unique properties of the instance, which can be then used, for example, to solve Algorithm Selection Problem (ASP, see [13]), or to predict how hard it will be for a given algorithm to find a good solution to the problem.

One of the methods of analyzing FL is to define and calculate specific measures, characterizing the FL and as a result—an instance. There are many well-known measures, both problem-specific and generic ones, e.g.: fitness distribution [23], epistasis [15], ruggedness [30], or neutrality [22]. A comprehensive survey over FL measures can be found in [11, 14].

Despite the great potential of FL analysis, it is often hard to carry one out, due to enormous amount of raw data. As a solution to this problem, Ochoa [20] proposed Local Optima Networks (LON-s). LON provides a compressed representation of FL, by retaining only the information about local optima. This model has yet been successfully applied to various optimization problems, such as: QAP [12, 20, 28], TSP [18, 19], or NK [20, 29].

Because it is not possible to directly build LON for larger-sized instances, a LON sampling method must be used. Since usually more accurate sampling requires more calculation, the question naturally arises: *What is the minimum computational effort required to build a LON that is a sufficient representation of the instance?* While this kind of research has already been conducted [19, 20, 26], we believe there is still much more to be done—FL analysis is often still too computationally demanding to be practically applicable.

## 2 Problem Definition

In the paper we investigate a cyclic manufacturing system, researched earlier in works [1]. This section provides comprehensive description of the problem with some properties proven.

## 2.1 Cyclic Assignment Problem in Production Cell

Let us consider a production cell consisting of multiple machines and a single operator. In the cell, in fixed time intervals (called cycle time), there are batches of products manufactured. Since the goods can be produced on many different machines, an operation-to-machine assignment must be devised. Because there is only one operator in the system, at any time, there can be at most one operation performed. Cyclic Assignment Problem in Production Cell (CAPPC) is a problem of finding such an assignment, production schedule and cycle time that the cycle time is minimal.

CAPPC can be formally defined as follows. There are  $o$  operations to be performed in the production cell, constituting a set  $\mathcal{O} = \{1, 2, \dots, o\}$ . The operations must be performed in the specific order,  $1 \rightarrow 2 \rightarrow \dots \rightarrow o$ , on the machines identified by the numbers from the set  $\mathcal{M} = \{1, 2, \dots, m\}$  (in this paper, we assume that  $m = 2$ ). The operation-to-machine assignment is determined by a tuple

$$P = (P(1), P(2), \dots, P(o)),$$

where  $P(i)$ ,  $i \in \mathcal{O}$  denotes the machine the operation  $i$  is to be performed on; and  $\mathcal{P}$  is a set of all possible assignments. Each operation  $i \in \mathcal{O}$  must be being performed uninterruptedly for  $p_i^{P(i)}$  time units. Moreover, between each two operations  $i, j \in \mathcal{O}$  performed one after another on the same machine  $a = P(i) = P(j)$ , there is an Uninterruptible setup with the duration of  $s_{i,j}^a$  time units.

**Definition 4.1** Minimal Part Set (MPS) is a set of copies of operations from the set  $\mathcal{O}$ . MPS-es are numbered by successive natural numbers  $x = 1, 2, \dots$

MPS-es are performed one after another, potentially an infinite number of times. A schedule for CAPPC is defined, for each MPS, by a pair of vectors

$$S^x = (S_1^x, S_2^x, \dots, S_o^x), \quad (1)$$

$$C^x = (C_1^x, C_2^x, \dots, C_o^x), \quad (2)$$

where  $x = 1, 2, \dots$  is a number of MPS. A schedule is feasible if and only if, for each MPS  $x \in \mathbb{N}^+$ , an assignment  $P \in \mathcal{P}$  and a cycle time  $T$ , the following constraints are satisfied:

1. At any time, there can be at most one operation or setup performed.
2. Setups and operations must not be interrupted.
3. Each operation  $i \in \mathcal{O}$  must be performed on the machine  $P(i) = a$  for  $p_i^a$  time.
4. Between each two operations  $i, j \in \mathcal{O}$  performed in a successive manner on the same machine  $a \in \mathcal{M}$ , there must be a setup  $s_{i,j}^a$  performed.
5. Each operation must be performed every cycle time  $T$ .
6. On each machine, before performing any operation from  $x + 1$ th MPS, all the operations from  $x$ th MPS must be finished.

The constraints can be formally defined as

$$\forall i \in \mathcal{O} \setminus \{o\} \forall x \in \mathbb{N}^+ \left( S_{i+1}^x x \geq S_i^x + p_i^{P(i)} + s(P, i + 1) \right), \quad (3)$$

$$\forall i \in \mathcal{O} \quad \forall x \in \mathbb{N}^+ \left( S_i^{x+1} x = S_i^x + T \right), \quad (4)$$

$$\forall i \in \mathcal{O} \quad \forall x \in \mathbb{N}^+ \left( C_i^x x = S_i^x + p_i^{P(i)} \right), \quad (5)$$

$$\forall x \in \mathbb{N}^+ \left( S_1^{x+1} x \geq C_o^x + s(P, 1) \right), \quad (6)$$

where  $s(P, i)$  is the setup performed before an operation  $i$  for a solution  $P$ . The setup time can be calculated from equation

$$s(P, i) = \begin{cases} S_{\max \mathcal{O}(P, i)}^{P(i)} & \text{for } |\mathcal{O}(P, i)| \geq 1, \\ S_{\max\{j \in \mathcal{O} : P(i) = P(j)\}}^{P(i)} & \text{otherwise,} \end{cases} \quad (7)$$

$$\mathcal{O}(P, i) = \{j \in \mathcal{O} : (j < i \wedge P(i) = P(j))\}. \quad (8)$$

CAPPC boils down to determining such an assignment, cycle time and acceptable schedule that the cycle time is minimal. In the context of solving algorithms, such CAPPC formulation may be unfavorable, due to the need to determine optimal values of multiple parameters:  $P, T, S, C$ . Therefore, in the further part of this sub-chapter, it will be shown how to determine the optimal schedule and cycle time for any assignment—so that the CAPPC can be formulated as the problem of determining the optimal assignment.

**Definition 4.2** Let  $P \in \mathcal{P}$  be a given assignment. Minimal cycle time  $T(P)$  is the minimal value of a cycle time for which at least one feasible schedule exist.

For a given assignment  $P \in \mathcal{P}$ , let us consider a feasible schedule, minimizing a cycle time  $T$  for any MPS  $x \in \mathbb{N}^+$ . From Eqs. (4) and (6), we have

$$T = S_1^{x+1} - S_1^x \geq C_o^x + s(P, 1) - S_1^x. \quad (9)$$

Now, the value of the right side of the inequity (9) must be determined. Let  $S_i^x(P)$ ,  $i \in \mathcal{O}$ , be the earliest time an operation  $i$  can be started. Then,

$$\begin{aligned} S_i^x(P) &= S_{i-1}^x(P) + p_{i-1}^{P(i-1)} + s(P, i) = \\ &= S_{i-2}^x(P) + p_{i-2}^{P(i-2)} + s(P, i-1) + p_{i-1}^{P(i-1)} + s(P, i) = \\ &= S_1^x + p_1^{P(1)} + p_2^{P(2)} + \dots + p_{i-1}^{P(i-1)} + s(P, 2) + s(P, 3) + \dots + s(P, i) = \\ &= S_1^x + \sum_{j=1}^{i-1} p_j^{P(j)} + \sum_{j=2}^i s(P, j). \end{aligned} \quad (10)$$

Defining  $C_i^x(P)$  in an analogous way to  $S_i^x(P)$  and substituting Eq. (10) into Eq. (5)

$$\begin{aligned}
C_i^x(P) &= S_i^x + p_i^{P(i)} = \\
&= S_1^x + \sum_{j=1}^{i-1} p_j^{P(j)} + \sum_{j=2}^i s(P, j) + p_i^{P(i)} = \\
&= S_1^x + \sum_{j=1}^i p_j^{P(j)} + \sum_{j=2}^i s(P, j). \tag{11}
\end{aligned}$$

Therefore, for any given  $P \in \mathcal{P}$

$$T \geq C_o^x(P) + s(P, 1) - S_1^x = \sum_{j=1}^o \left( p_j^{P(j)} + s(P, j) \right), \tag{12}$$

and thus, a minimal cycle time  $T(P)$  equals

$$T(P) = \sum_{j \in \mathcal{O}} \left( p_j^{P(j)} + s(P, j) \right). \tag{13}$$

Equation (13) allows us to rewrite CAPPCC into the problem of finding any assignment  $P^* \in \mathcal{P}$ , that a minimal cycle time  $T(P^*)$  is minimal

$$P^* \in \arg \min_{P \in \mathcal{P}} \{T(P)\}. \tag{14}$$

## 2.2 CAPPCC in Non-permutational FSP

The considered problem is a non-permutational Flow Shop Problem (FSP) with CAPPCC-cells instead of the machines (CFSPCC). In FSP, there is a set of jobs  $\mathcal{J} = \{1, 2, \dots, n\}$  to be performed. Each jobs consists of  $q$  operations to be performed in a predefined order

$$(i-1)q+1 \rightarrow (i-1)q+2 \rightarrow \dots \rightarrow iq,$$

on the machines

$$1 \rightarrow 2 \rightarrow \dots \rightarrow q,$$

constituting a set  $\mathcal{Q} = \{1, 2, \dots, q\}$  (here the machines are replaced by the production cells). The order in which the operations are performed in the cells is determined by a n-tuple

$$\pi = (\pi_1, \pi_2, \dots, \pi_q), \tag{15}$$

where

$$\pi_i = (\pi_i(1), \pi_i(2), \dots, \pi_i(n)), \quad i \in \mathcal{J}, \quad (16)$$

is the order of operations performed in cell  $i$ .

**Lemma 4.1** *Let  $\pi$  be, for a given CFSPPC instance, the order in which the operations are to be performed, and  $T_i$ —the minimal cycle time of the CAPPCC instance constructed from the operations  $\pi_i(1), \pi_i(2), \dots, \pi_i(q)$ . Then*

$$T = \max\{T_i : i \in \mathcal{Q}\}$$

is the minimal cycle time of the CFSPPC instance for  $\pi$ .

*Proof* We will prove that (\*) there is at least one feasible schedule for  $T$  and (\*\*) there is no feasible schedule for any  $T' < T$ . Let  ${}^i S_j^x$  and  ${}^i C_j^x$  denote, respectively, the moment an operation  $j + (i - 1)q$  is started and the finished, according to an optimal schedule for the CAPPCC instance created from cell  $i$ . We will prove (\*) by showing that schedule

$$S^x = \begin{pmatrix} {}^1 S_1^x, & {}^1 S_2^x, & \dots, & {}^1 S_n^x, \\ T + {}^2 S_1^x, & T + {}^2 S_2^x, & \dots, & T + {}^2 S_n^x, \\ \vdots & \vdots & \ddots & \vdots \\ (n-1)T + {}^q S_1^x, & (n-1)T + {}^q S_2^x, & \dots, & (n-1)T + {}^q S_n^x \end{pmatrix}, \quad (17)$$

$$C^x = \begin{pmatrix} {}^1 C_1^x, & {}^1 C_2^x, & \dots, & {}^1 C_n^x, \\ T + {}^2 C_1^x, & T + {}^2 C_2^x, & \dots, & T + {}^2 C_n^x, \\ \vdots & \vdots & \ddots & \vdots \\ (n-1)T + {}^q C_1^x, & (n-1)T + {}^q C_2^x, & \dots, & (n-1)T + {}^q C_n^x \end{pmatrix}, \quad (18)$$

is feasible. From the assumptions, the schedule  $S^x, C^x$ , satisfies the CAPPCC constraints because

$$\forall i \in \mathcal{Q} \quad (T_i \leq T). \quad (19)$$

Now, let us show that the schedule also preserves the technological order,

$$\forall i \in \mathcal{J} \quad \forall j \in \{(i-1)q+1, (i-1)q+2, \dots, iq\} \quad (C_j^x \leq S_{j+1}^x). \quad (20)$$

Let  $k = (j - 1) \bmod n + 1$  and  $l = \lceil j/n \rceil$ . Then



$$\begin{aligned}
C_j^x &= (k-1)T + {}^k C_l^x \leq (k-1)T + {}^k S_1^x + T = \\
&= kT + {}^k S_1^x = \\
&= kT + {}^{k+1} S_1^x \leq kT + {}^{k+1} S_l^x = \\
&= S_{j+1}^x,
\end{aligned} \tag{21}$$

which proofs (\*). The proof of (\*\*) is trivial. Assume that  $T' < T$  and there exist a feasible schedule for  $T'$ . Then, there also exist a feasible schedule for a cell

$$i \in \arg \max_{j \in \mathcal{Q}} T_j,$$

for  $T'$ . It leads to a contradiction, since  $T' < T = T_i$  and  $T_i$  is the minimal cycle time for that cell. ■

Based on Lemma 4.1, any instance of CFSPPC can be divided into  $q$  independent, one-cell optimization subproblems. As shown in [1], such approach can be used to devise an efficient heuristic algorithm. Therefore we will study more closely the properties of the one-cell CFSPPC (referred to later as Cyclic Assignment and Scheduling Problem in Production Cell, CASPPC).

### 2.3 Cyclic Assignment and Scheduling Problem in Production Cell

Since in CASPPC there is only one cell (as shown in Fig. 1), the order of operation performance can be described by a single tuple

$$\pi = (\pi(1), \pi(2), \dots, \pi(n)), \quad n = o, \tag{22}$$

instead of a tuple of tuples, defined in Eqs. (15) and (16). Let  $\Pi$  be the set of all possible orders of operations. CASPPC is a problem of finding such  $\pi \in \Pi$ ,  $P \in \mathcal{P}$  and  $T$ , that at least one feasible schedule exists and the cycle time is minimal. Because operations can be renumbered, the formulation can be simplified into finding such  $\pi$  and  $P$  that the minimal cycle time  $T_\pi(P)$  of the instance of CAPPCC constructed by the  $\pi$  is minimal. In [1] a polynomial-time exact algorithm was proposed for CAPPCC, allowing to further narrow the search space

$$\pi^* \in \arg \min_{\pi \in \Pi} \left\{ \min_{P \in \mathcal{P}} \{T_\pi(P)\} \right\}, \tag{23}$$

since

$$\min_{P \in \mathcal{P}} \{T_\pi(P)\} \tag{24}$$

can be calculated in  $O(o^3)$  for  $m = 2$ .

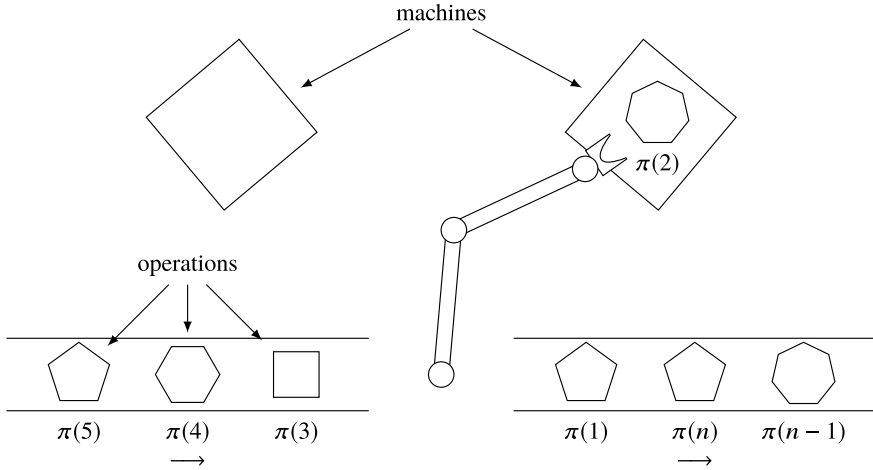


Fig. 1 Cyclic production cell

### 3 Local Optima Networks

Local Optima Network (LON) is a compact representation of a Fitness Landscape, further embedding the properties of search algorithms. In this section we will introduce some basic concepts regarding both FL and LON.

#### 3.1 Definitions

FL describes a solution space of a problem from the perspective of a local search algorithm. Work [27] defines FL as a triple  $(S, V, f)$ , where:

- $S$  is a solution space. In the researched problem  $S = \Pi, |\Pi| = (n - 1)!$ .
- $V$  is a neighborhood function,  $V : S \rightarrow \mathfrak{P}(S)$ , where  $\mathfrak{P}(S)$  is a powerset of  $S$ . The definition of the neighborhood is taken from [1].
- $f$  is an objective (fitness) function,  $f : S \rightarrow \mathbb{R}$ . In the researched problem

$$f(\pi) = \min_{P \in \mathcal{P}} \{T_{\pi}(P)\}.$$

Based on the notation introduced above, LON can be formally defined as a directed graph  $(N, E)$ , where  $N$  is a set of nodes and  $E$  is a set of edges. Each node from  $N$  represents a local minimum, i.e. such solution  $\pi \in \Pi$ , that

$$\forall \pi' \in V(\pi) \quad (f(\pi') \geq f(\pi)). \tag{25}$$

Therefore

$$N = \left\{ \pi \in \Pi : \forall \pi' \in V(\pi) \quad (f(\pi') \geq f(\pi)) \right\}. \quad (26)$$

Ochoa et al. [20] proposed two definitions of edges for LON: *basin-transition* and *escape edges*. In this paper we have chosen *basin-transition edges* as their weights can be computed easier. This definition is also used e.g. in [3, 20]. Edge  $(\pi', \pi'')$  exists in  $E$  if and only if  $\pi''$  can be obtained by applying kick operator (here the operator is defined as applying a random move twice) to the solution  $\pi'$ , followed by a steepest descend algorithm run. This procedure mimics diversification strategies used in heuristic algorithms. Let  $g : \Pi \rightarrow \Pi$  be a function assigning to any  $\pi \in \Pi$  a result of the steepest descend algorithm (shown in Algorithm 1) and

$$V^k(\pi) = \bigcup_{\pi' \in V(\pi)} V(\pi'), \quad (27)$$

a set of solutions obtained by applying all possible kick operators to  $\pi$ . Then, the set  $E$  can be defined as

---

**Algorithm 1:** Steepest descend algorithm for CASPPC

---

**Input** : Initial solution  $\pi_{init} \in \Pi$

**Output**: Local minimum  $\pi \in \Pi$

```

1  $\pi' \leftarrow \pi_{init}$ ;
2 repeat
3    $\pi \leftarrow \pi'$ ;
4    $V_{min} \leftarrow \arg \min_{\pi' \in V(\pi)} f(\pi')$ ;
5   Sort elements from  $V_{min}$  in lexicographic order and put the first element in  $\pi'$ ;
6 until  $f(\pi) > f(\pi')$ ;
7 return  $\pi$ 

```

---

$$E = \left\{ (\pi', \pi'') : \pi' \in N \wedge \pi'' \in \{g(\pi) : \pi \in V^k(\pi')\} \right\}. \quad (28)$$

A weight  $d$  of an edge  $(\pi', \pi'')$  represents the probability of transition from solution  $\pi'$  to  $\pi''$  by a local search algorithm, while trying to escape the local minimum  $\pi'$ . The weight function is defined as

$$d((\pi', \pi'')) = \frac{|\{\pi \in V^k(\pi') : g(\pi) = \pi''\}|}{|V^k(\pi')|}. \quad (29)$$

Since weights correspond to the probabilities, no weight can be larger than one

$$\forall \varepsilon \in E \quad (d(\varepsilon) \leq 1), \quad (30)$$

and the sum of weights of edges originating from any node equals one

$$\forall \pi' \in N \left( \sum_{\pi'' \in \{\pi : (\pi', \pi) \in E\}} d((\pi', \pi'')) = 1 \right). \quad (31)$$

### 3.2 LON Measures

Local Optima Networks can be described by various measures. In this paper we investigate some well-known ones:

- **assortativity**—measure of nodes preferences to be connected to nodes with similar parameters, based on o [17]. Let  $\pi \in N$  be a LON node. Then, following variants of node parameter  $x$  were tested:

- **assortativity-in**—a number of incoming edges

$$x_{\text{in}}(\pi) = |\{(\pi', \pi'') \in E : \pi'' = \pi\}|; \quad (32)$$

- **assortativity-out**—a number of outgoing edges

$$x_{\text{out}}(\pi) = |\{(\pi', \pi'') \in E : \pi' = \pi\}|; \quad (33)$$

- **assortativity-total**—a sum of a number of outgoing and incoming edges

$$x_{\text{total}}(\pi) = x_{\text{in}}(\pi) + x_{\text{out}}(\pi); \quad (34)$$

- **assortativity-bin**—a size of a node attraction basin

$$x_{\text{bin}}(\pi) = |\pi' \in \Pi : g(\pi') = \pi|; \quad (35)$$

- **assortativity-of**—an objective function of the node

$$x_{\text{of}}(\pi) = C_{\text{max}}(\pi). \quad (36)$$

Formally, the assortativity is defined as:

$$A = \frac{\sum_i e_{i,i} - \sum_i \left( \sum_j e_{i,j} \sum_j e_{j,i} \right)}{1 - \sum_i \left( \sum_j e_{i,j} \sum_j e_{j,i} \right)}, \quad (37)$$

where  $e_{i,j}$  is the fraction of edges connecting nodes of types  $i$  and  $j$ , e.g. such edges  $(\pi', \pi'') \in E$ , that  $x(\pi') = i$  and  $x(\pi'') = j$ .

- **global clustering** (transitivity)—a measure indicating the likeliness that if edges  $(a, b)$  and  $(b, c)$  exist, the edge  $(a, c)$  also exists. Nodes  $a, b, c$  constitute a triangle. Clustering can be formally defined [16] as

$$C = 3 \times \frac{\text{number of triangles}}{\text{number of triples}}, \quad (38)$$

where “triple” denotes a node with edges running to an unordered pair of other nodes. The computational complexity of the implementation equals  $O(|N| \langle k \rangle^2)$ .

- **average shortest path to optimum**—the average shortest path length from every node to the closest global optimum.
- **minimum as percent**—the percentage ratio of LON nodes with the minimum value of the objective function (global optima)

$$O = \frac{\left| \arg \max_{\pi \in N} C_{max}(\pi) \right|}{|N|} \cdot 100\% \quad (39)$$

The implementation for measures was provided by `graph-tool` library [21].

## 4 Computational Experiments

Analyzing the entire FL for the instance of a larger size is impossible in practice. Therefore, sampling methods are used, which—in principle—allow to create a representative “snapshot” of the solution space. The need to determine how such procedure affects the obtained information is obvious. For this purpose, we will generate complete FLs of small-sized instances, calculate the exact values of the measures, and then simulate sampling procedure.

### 4.1 Experimental Setup

The experiments were conducted on random instances, of the sizes ranging from 6 to 9 operations. For each size, 30 instances were generated. The FLs of the instances were sampled in steps, called *snapshots*. For a given instance, in a snapshot  $S_c$ , exactly  $S_c$  percent of solutions (selected randomly) from the entire search space are considered. In order to quantify the impact of sampling the search space on the values of LON measures, a relative measure deviation concept is introduced. Let  $M_i(S_c)$  be a value of the relative measure deviation of a measure  $i$ , for a snapshot  $S_c$ :

$$\Delta M_i(S_c) = \frac{M_i(100\%) - M_i(S_c)}{M_i(100\%)}. \quad (40)$$

For each measure and instance, relative measure deviations were calculated for snapshots  $S_c = 5, 10, \dots, 95\%$ .

## 4.2 Results

In this section, results of computational experiments (presented in Figs. 2, 3, 4, 5, 6, 7, 8 and 9) will be presented and discussed.

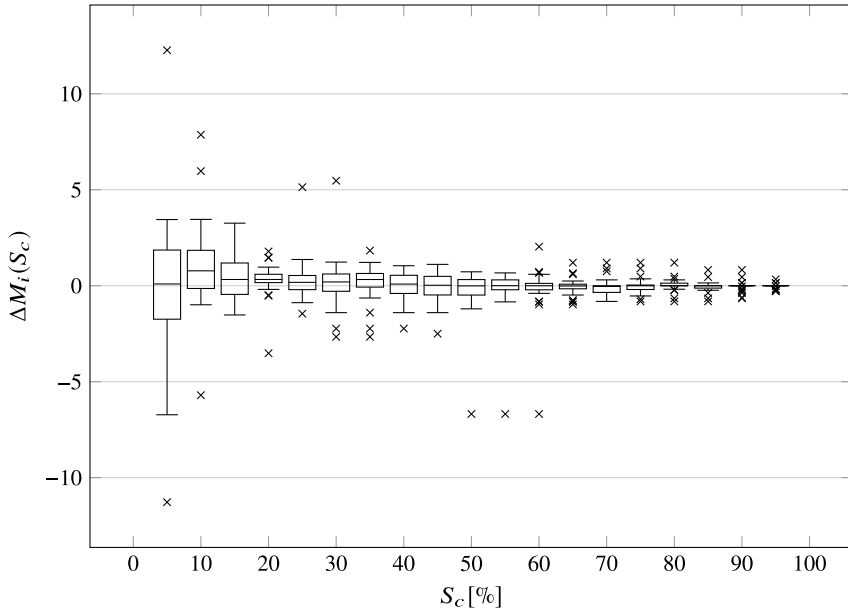
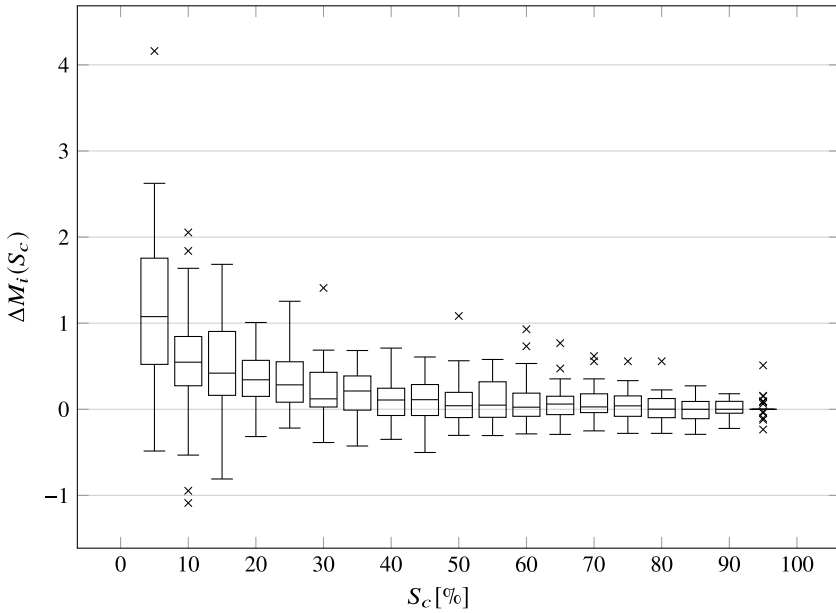
### Assortativity-in

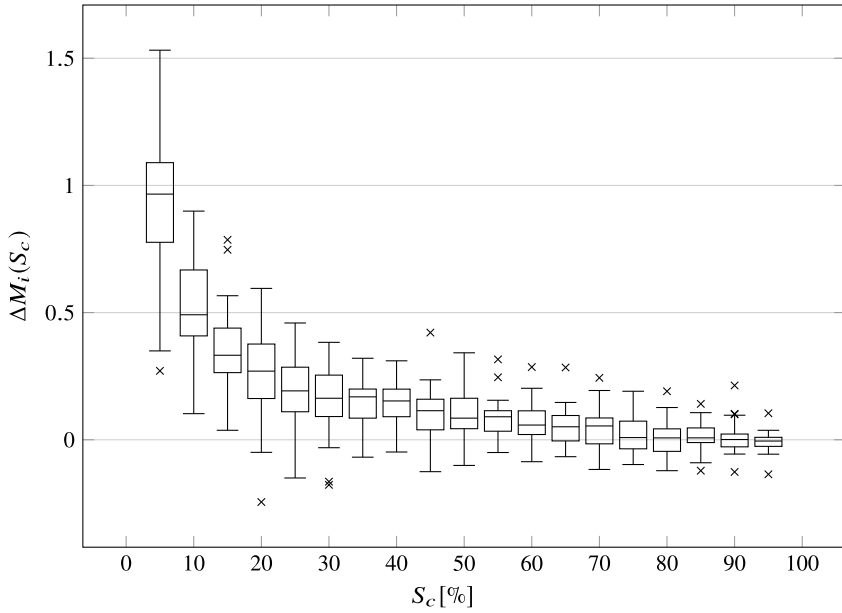
First, we will describe the data gathered on assortativity-in. The relative deviations of the considered measure values, for different snapshots and instance sizes, are shown on Figs. 2 and 3.

For the smallest instances with  $n = 6$  (shown on Fig. 2a), the median deviation of the measure remains between 0 and 1. It starts with the value close to 0, when only 5% of solution space is sampled ( $S_c = 5\%$ ). Then, when  $S_c = 10\%$ , there is sudden change of value to 1, with a gradual decrease afterwards, until reaching values close to 0 by  $S_c = 45\%$ . This predictable behaviour can be observed to even greater extent for larger instances. The interquartile range is relatively large (more than 2 standard deviations) for small  $S_c$ , and decreases with the increasing percentage of solution searched. For  $S_c = 15\%, \dots, 60\%$ , the range stabilizes at about 1 relative deviation, then a next drop can be seen. Empirical distributions of the measure deviations have different shapes—some are similar to a normal distribution, while other are of a significant skewness (seen as long, asymmetric whiskers). The data is rather heterogeneous, multiple outliers are present, ranging up to 6–10 relative deviations from 0.

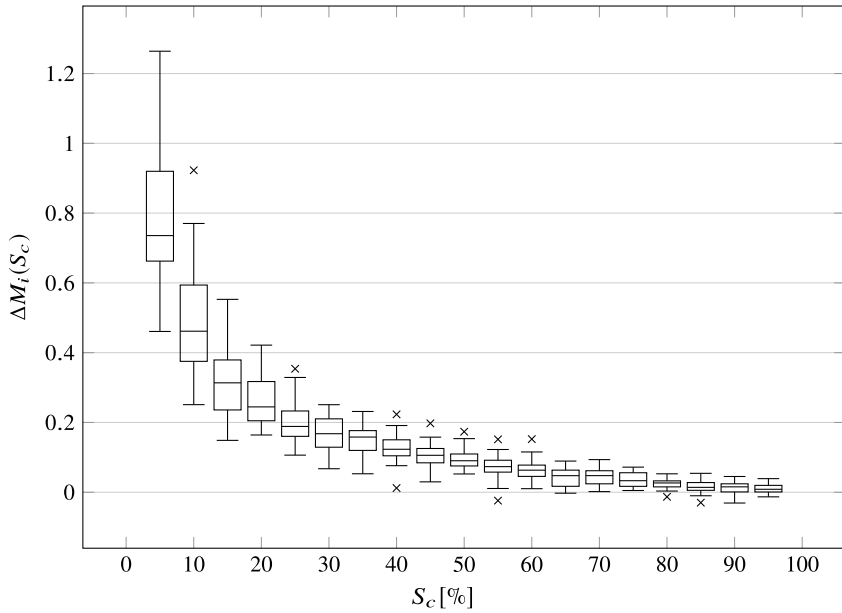
For the instances with  $n = 7$  operations (Fig. 2b), the most obvious difference is lower median measure deviations, while the trend of decreasing its value with the increase of  $S_c$  remains. The number of outliers, although lower, is still significant. The deviation empirical distributions is, again, similar to normal, with occasional larger skewness.

The next analyzed figure, Fig. 3a, shows the relative deviation of assortativity-in measured for the instances of the size of  $n = 8$ . The median deviation is further decreased, with a very clearly visible downward trend. The same downward tendency can be observed for the measure deviation variance. The trend of decreasing both measure deviation median and variance continues for the instance size  $n = 9$  (shown on Fig. 3b).

(a) Instance size  $n = 6$ (b) Instance size  $n = 7$ **Fig. 2** Relative assortativity-in deviation for different snapshots



(a) Instance size  $n = 8$



(b) Instance size  $n = 9$

Fig. 3 Relative assortativity-in deviation for different snapshots



## Other Measures

We will not present such detailed description of the experiments performed for the other metrics (Figs. 5, 6, 7, 8 and 9). The changes in the deviations of metrics, such as: assortativity-out, assortativity-total, assortativity-of, global clustering, average shortest path to optimum or minimum as percent are similar to the ones described for assortativity-in. However, several abnormalities are worth noting. For assortativity-of and instance size  $n = 6$  (Fig. 4a), the outliers, especially for small snapshots, lie much further from 0, then for any other measure. Therefore, this measure proved to be susceptible to the undersampling issues for smaller problem sizes. Another interesting observation can be made for the average shortest path to optimum for  $n = 6$  (Fig. 4b). The outliers for this experiments tend to assume common, “discrete” values. It is a result of a small instances size and therefore—LONs to be measured are small graphs with relatively short paths possible.

## Discussion

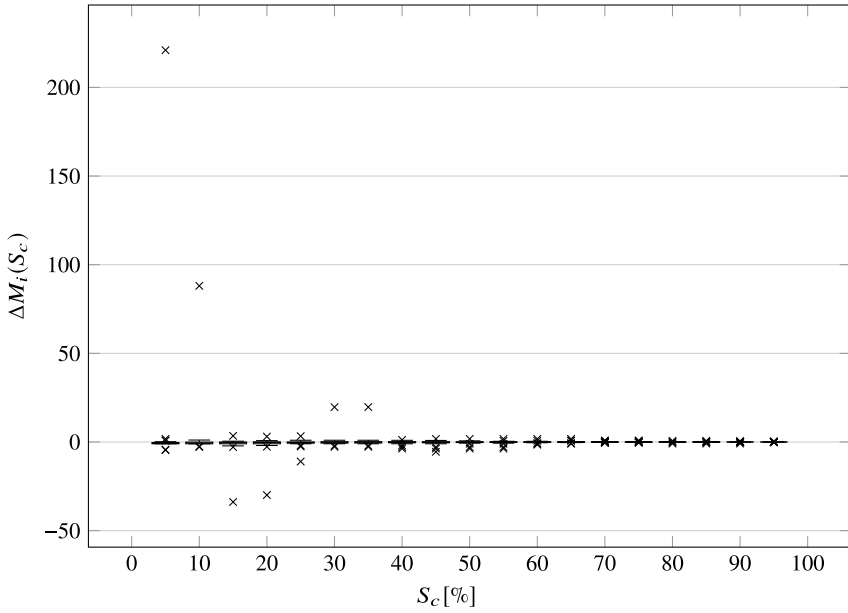
The experiments carried out indicate, that the variability of the values of the measures is not only related to  $S_c$ , but also to the size of the instance. As the size of instance increases, for a fixed  $S_c$ , the variance decreases. This fact suggests, that the computational effort necessary to obtain the value of a measure with a given error tolerance increases more slowly than the size of the solution space. Nevertheless, the determination of whether the polynomial relation used, among others, in [12], is sufficient for a problem with exponential solution space, requires further research.

In addition to the significant variability of the measure, the average deviation of the measure is non-zero (for most measures the value is undervalued). Hence, the expected value of measure value is influenced not only by the specificity of the instance itself (intentional behavior), but also by the computational effort to sample the solution space. This dependence seems to differ from measure to measure and is especially evident for larger instance sizes (and therefore—for the experiments with more samples taken). Specifying the form of this dependency could allow for applying an appropriate amendment and unbiased measure estimation.

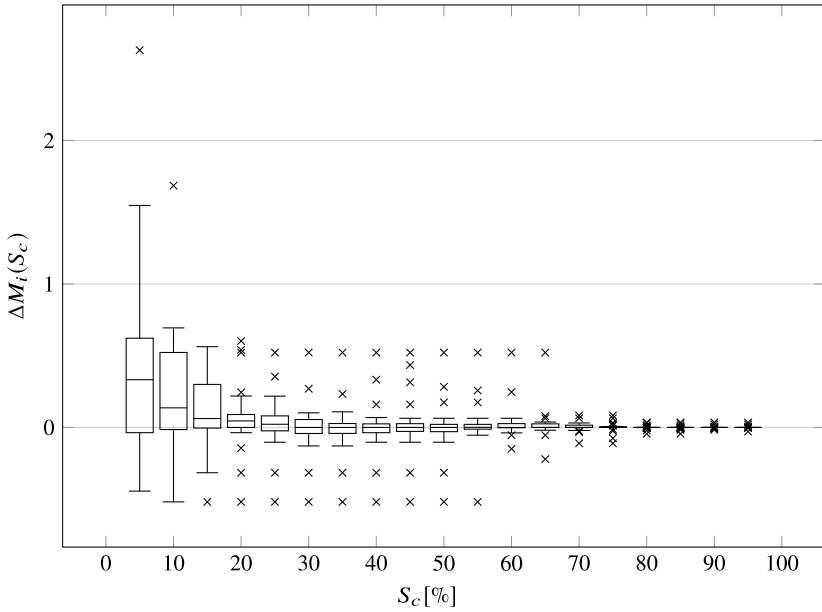
## 5 Remarks and Conclusions

The paper examines the impact of sampling the solution space on the values of selected measures used in fitness landscape analysis. The examined measures proved to be susceptible to insufficient sampling effort. With the increasing of the number of samples, not only their variance but also the expected value changed.

The results obtained relate only to instances of a small size, hence the natural next step in the research will be experiments for the instances of sizes found in benchmarks (such as, for example, Taillard instances). An interesting challenge is to



(a) Assortativity-of for instance size  $n = 6$



(b) Average shortest path to optimum for instance size  $n = 6$

**Fig. 4** Examples of interesting anomalies in the proposed metrics values

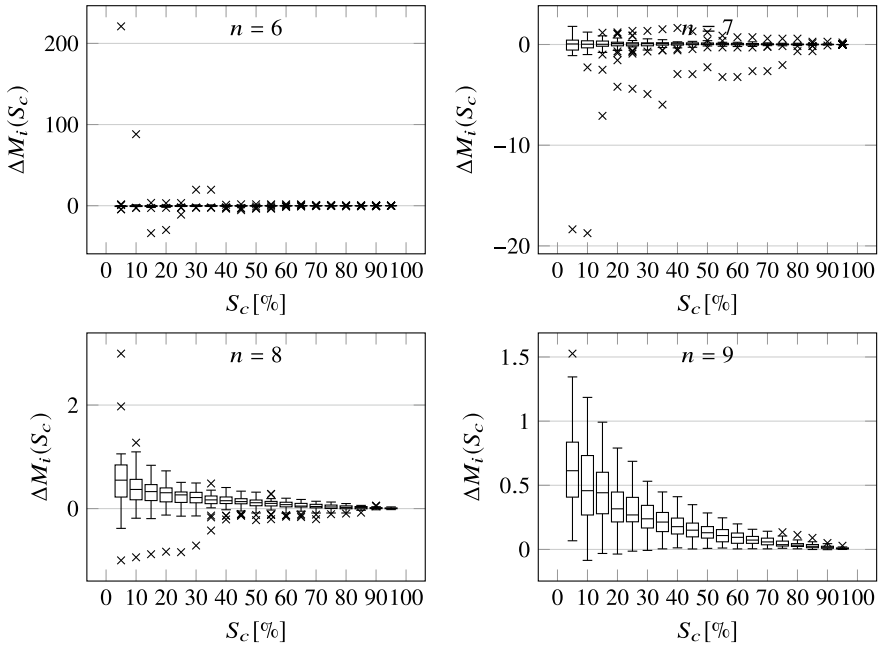


Fig. 5 Boxplots of assortativity-of and different instance sizes

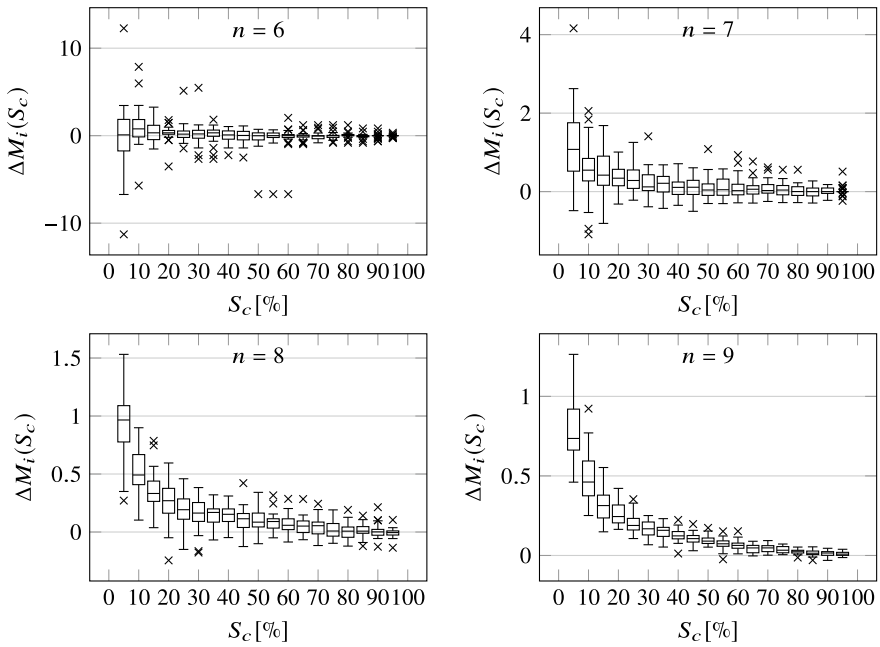


Fig. 6 Boxplots of assortativity-in and different instance sizes

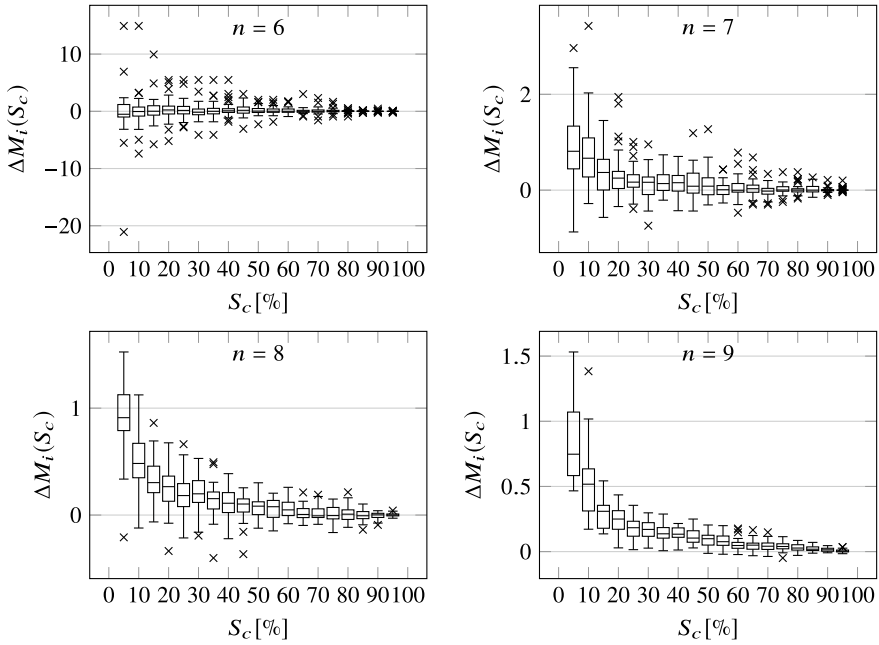


Fig. 7 Boxplots of assortativity-out and different instance sizes

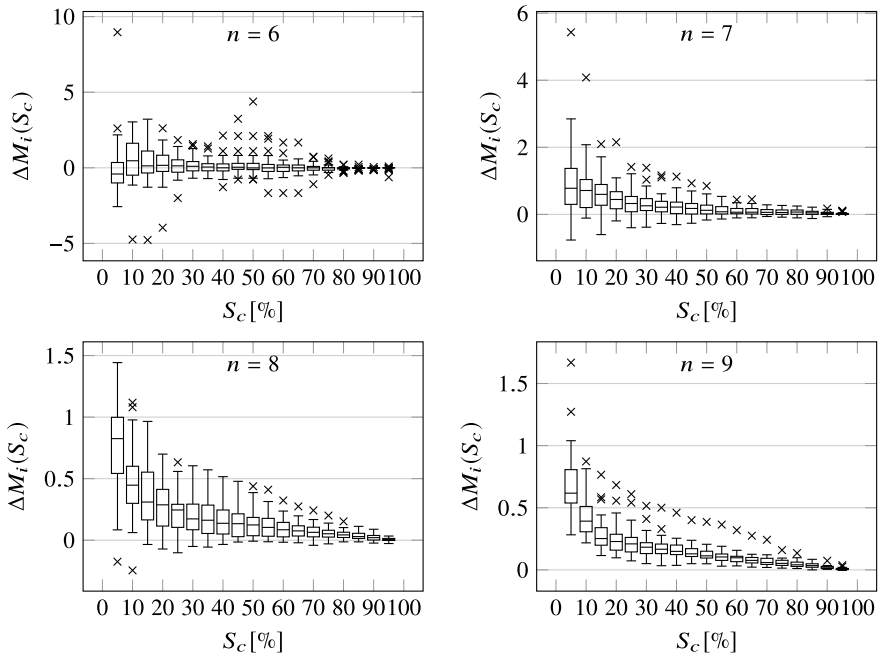
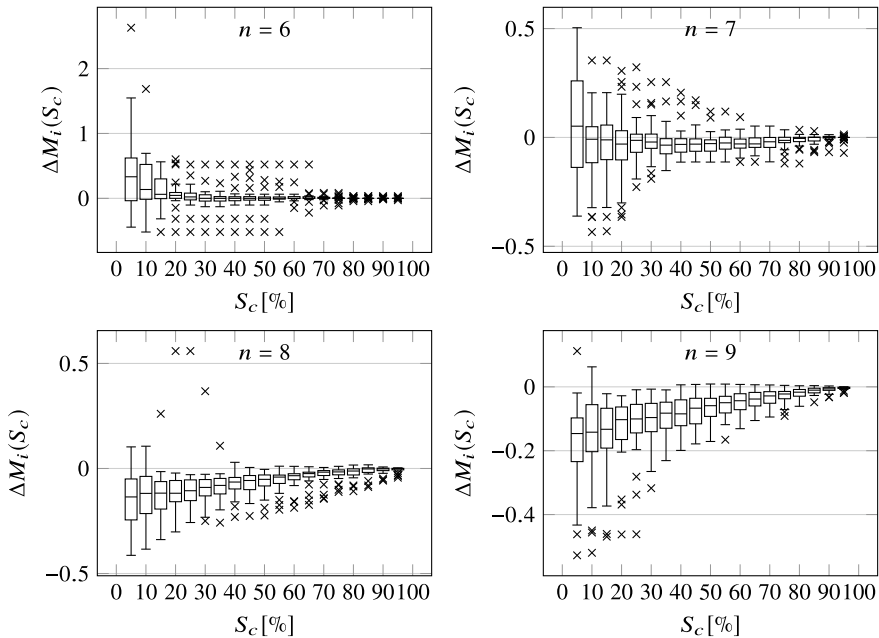


Fig. 8 Boxplots of assortativity-bin and different instance sizes



**Fig. 9** Boxplots of average shortest paths to closest local optimum and different instance sizes

determine the scale of variability of the measures, both in the context of a constant and a variable computational effort—potentially leading to unbiased estimation of the measure values, with minimal computational effort.

**Acknowledgements** The chapter was partially supported by the National Science Centre of Poland, grant OPUS number DEC 2017/25/B/ST7/02181.

## References

1. Bożejko, W., Gnatowski, A., Idzikowski, R., Wodecki, M.: Cyclic flow shop problem with two-machine cells. *Arch. Control. Sci.* **27**(2), 151–167 (2017)
2. Bożejko, W., Gnatowski, A., Klempous, R., Affenzeller, M., Beham, A.: Cyclic scheduling of a robotic cell. In: 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), pp. 379–384. IEEE (2016)
3. Bożejko, W., Gnatowski, A., Niżyński, T., Affenzeller, M., Beham, A.: Local optima networks in solving algorithm selection problem for TSP. In: Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., Kacprzyk, J. (eds.) *Contemporary Complex Systems and Their Dependability. DepCoS-RELCOMEX 2018. Advances in Intelligent Systems and Computing*, vol. 761, pp. 83–93. Springer, Cham (2019)
4. Bożejko, W., Pempera, J., Wodecki, M.: Minimal cycle time determination and golf neighborhood generation for the cyclic flexible job shop problem. *Bull. Pol. Acad. Sci.: Tech. Sci.* **66**(3), 333–344 (2018)

5. Chaudhry, I.A., Khan, A.A.: A research survey: Review of flexible job shop scheduling techniques. *Int. Trans. Oper. Res.* **23**(3), 551–591 (2016)
6. Crama, Y., Kats, V., van de Klundert, J., Levner, E.: Cyclic scheduling in robotic flowshops. *Ann. Oper. Res.* **96**(1), 97–124 (2000)
7. Dawande, M., Geismar, H.N., Sethi, S.P., Sriskandarajah, C.: Sequencing and scheduling in robotic cells: recent developments. *J. Sched.* **8**(5), 387–426 (2005)
8. Dawande, M.W., Geismar, H.N., Suresh, P.S., Sriskandarajah, C., Sethi, S., Sriskandarajah, C.: *Throughput Optimization in Robotic Cells*. Springer, Boston (2007)
9. Gultekin, H., Coban, B., Akhlaghi, V.E.: Cyclic scheduling of parts and robot moves in m-machine robotic cells. *Comput. Oper. Res.* **90**, 161–172 (2018)
10. Hall, N.G., Kamoun, H., Sriskandarajah, C.: Scheduling in robotic cells: classification, two and three machine cells. *Oper. Res.* **45**(3), 421–439 (1997)
11. Humeau, J., Liefvooghe, A., Talbi, E.G., Verel, S.: ParadisEO-MO: From fitness landscape analysis to efficient local search algorithms. Technical report RR-7871, INRIA. <https://hal.inria.fr/hal-00665421v2>. Accessed 30 March 2019
12. Iclanzan, D., Daolio, F., Tomassini, M.: Data-driven local optima network characterization of QAPLIB instances. In: *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pp. 453–460. ACM Press, New York (2014)
13. Kotthoff, L.: Algorithm selection for combinatorial search problems: a survey. In: Bessiere, C., De Raedt, L., Kotthoff, L., Nijssen, S., O’Sullivan, B., Pedreschi, D. (eds.) *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, pp. 149–190. Springer International Publishing, Cham (2016)
14. Lu, H., Shi, J., Fei, Z., Zhou, Q., Mao, K.: Measures in the time and frequency domains for fitness landscape analysis of dynamic optimization problems. *Appl. Soft Comput.* **51**, 192–208 (2017)
15. Naudts, B., Suys, D., Verschoren, A.: Epistasis as a basic concept in formal landscape analysis. In: *Proceedings of the 7th International Conference on Genetic Algorithms*, pp. 65–72. Morgan Kaufmann, Burlington (1997)
16. Newman, M.: The structure and function of complex networks. *SIAM Rev.* **45**(2), 167–256 (2003)
17. Newman, M.E.J.: Mixing patterns in networks. *Phys. Rev. E* **67**, 026126 (2003)
18. Ochoa, G., Veerapen, N.: Additional dimensions to the study of funnels in combinatorial landscapes. In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference—GECCO’16*, pp. 373–380. ACM Press, New York (2016)
19. Ochoa, G., Veerapen, N.: Mapping the global structure of TSP fitness landscapes. *J. Heuristics* **24**(3), 265–294 (2017)
20. Ochoa, G., Verel, S., Daolio, F., Tomassini, M.: Local optima networks: a new model of combinatorial fitness landscapes. In: Richter, H., Engelbrecht, A. (eds.) *Recent Advances in the Theory and Application of Fitness Landscapes*, pp. 233–262. Springer, Berlin (2014)
21. Peixoto, T.P.: The graph-tool python library. [http://figshare.com/articles/graph\\_tool/1164194](http://figshare.com/articles/graph_tool/1164194) (2014). Accessed 30 March 2019
22. Reidys, C.M., Stadler, P.F.: Neutrality in fitness landscapes. *Appl. Math. Comput.* **117**(2–3), 321–350 (2001)
23. Rosé, H., Ebeling, W., Asselmeyer, T.: The density of states — a measure of the difficulty of optimisation problems. In: Guervós, J.J.M., Adamidis, P., Beyer, H.G., Schwefel, H.P., Fernández-Villacañas, J.L. (eds.) *Parallel Problem Solving from Nature — PPSN IV*. PPSN 1996. *Lecture Notes in Computer Science*, vol. 2439, pp. 208–217. Springer, Berlin (1996)
24. Ruiz, R., Vázquez-Rodríguez, J.A.: The hybrid flow shop problem. *Eur. J. Oper. Res.* **205**(1), 1–18 (2010)
25. Sethi, S.P., Sriskandarajah, C., Sorger, G., Blazewicz, J., Kubiak, W.: Sequencing of parts and robot moves in a robotic cell. *Int. J. Flex. Manuf. Syst.* **4**(3–4), 331–358 (1992)
26. Shirakawa, S., Nagao, T.: Bag of local landscape features for fitness landscape analysis. *Soft Comput.* **20**(10), 3787–3802 (2016)

27. Stadler, P.F.: Fitness landscapes. In: Lässig, M., Valleriani, A. (eds.) *Biological Evolution and Statistical Physics*, pp. 183–204. Springer, Berlin (2002)
28. Thomson, S.L., Ochoa, G., Daolio, F., Veerapen, N.: The effect of landscape funnels in QAPLIB instances. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion on—GECCO'17*, pp. 1495–1500. ACM Press, New York (2017)
29. Tomassini, M., Verel, S., Ochoa, G.: Complex-network analysis of combinatorial spaces: the NK landscape case. *Phys. Rev. E* **78**(6), 066114 (2008)
30. Vassilev, V.K., Fogarty, T.C., Miller, J.F.: Information characteristics and the structure of landscapes. *Evol. Comput.* **8**(1), 31–60 (2000)

# Cyclic Route Planning



# Coordination of Cyclic Motion Processes in Free-Ranging Multiple Mobile Robot Systems



Elzbieta Roszkowska 

**Abstract** We consider a Multiple Mobile Robot System (MMRS) viewed as a group of autonomous robots sharing a common 2D motion space. Each robot performs a mission that requires it to travel a number of times along a specific, independently planned closed path. The robots operate asynchronously and are able to control their motion with path-following algorithms that allow each of them to correctly perform its mission when alone on the stage. When sharing the motion space, the robots must refine their motion strategies in order to avoid collisions, through modification of their paths, velocity profiles or both. Following our earlier contributions, we represent MMRS as a class of RAS (Resource Allocation System) that abstracts in a discrete form the motion space and the motion processes of the robots. A model of the feasible dynamic behavior of the robot system is then obtained by mapping the distinguished RAS into a DFSA (Deterministic Finite State Automaton) that ensures collision avoidance among the robots. Based on this model, we formulate the deadlock avoidance problem, discuss its complexity, and demonstrate relevant algorithms to solve it. Finally, we propose a control architecture that implements the described control logic and combines it with the priority control, thus receiving a flexible controller for MMRS.

## 1 Introduction

The use of a mobile robot team in place of one robot substantially increases the performance of many robotic applications, including those related to transport, area searching, search and rescue, interplanetary exploration, extraction of minerals, or agriculture and forestry. A key issue in the design of such systems is to coordinate the movement of a number of robots operating in the same workspace. Regardless

---

E. Roszkowska (✉)

Department of Cybernetics and Robotics, Faculty of Electronics, Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland  
e-mail: [elzbieta.roszkowska@pwr.edu.pl](mailto:elzbieta.roszkowska@pwr.edu.pl)

© Springer Nature Switzerland AG 2020

W. Bożejko and G. Bocewicz (eds.), *Modelling and Performance Analysis of Cyclic Systems*, Studies in Systems, Decision and Control 241,  
[https://doi.org/10.1007/978-3-030-27652-2\\_5](https://doi.org/10.1007/978-3-030-27652-2_5)

of their tasks, the robots must be able to effectively share a common area in order to prevent the mutual disruption of traffic and effectively pursue their missions.

The prevailing approach to modeling Multiple Mobile Robot Systems (MMRS) consists in the abstraction of the dynamics of the robots in time, and considering the problem of the robots' coordination over time. Earlier works mostly concentrated on motion planning with respect to collision avoidance and performance optimization. According to [17], two categories of approaches to these problems—centralized and decoupled—can be distinguished as opposite ends of the spectrum of solutions. A centralized approach typically constructs a path in a composite configuration space, which is formed by the Cartesian product of the configuration spaces of the individual robots, e.g. [1]. A decoupled approach typically generates paths for each robot independently, and then a coordination diagram is used to plan a collision-free trajectory along the paths, e.g. [3]. Most often the problem of collision-free motion planning is decomposed into two subproblems: path planning and trajectory planning. Path planning finds geometric paths that do not intersect static obstacles, and trajectory planning determines how fast each robot must move along its path to avoid collision with others.

However, the control of a multiple robot system based solely on motion planning has a significant shortcoming. At the robot coordination level, the realization of motion plans is an open-loop control policy, based on deterministic time functions. Such a control is very sensitive to the system randomness, which, given the autonomous and asynchronous operation of the robots, makes the eventual applicability of these open-loop control plans highly questionable.

Therefore, more contemporary solutions use algorithms that calculate robot coordination decisions online, taking into account dynamic models of the robots and information about their current state. Two concepts representative for this approach are Reciprocal Collision Avoidance [16] and the Potential Fields [4]. However, although very effective locally, these methods cannot be easily adapted for the synthesis of the required global system behavior. The continuous-time abstraction used to describe a single robot, when applied to a multiple robot system yields solutions that are not scalable and do not capture the asynchronous character of the robot cooperation. In view of the above, a promising approach is the hybrid control concepts that combines a DES-based (Discrete Event System) supervisory control logic with a CTS-based (Continuous Time System) robot motion control. While various aspects of this type of approach have been recently considered, e.g., [5–8, 10], few works provide formal methodologies that are adaptable to changes in problem settings and guarantee the correct and efficient operation of MMRS in the entire domain of their model definition.

In this chapter, we consider a group of mobile robots, whose operation will be viewed as *a set of cyclic robot motion processes* concurrently executed in a shared area. A practical example of such a system can be a Flexible Assembly System (FAS), in which all of the parts that are needed to make one assembly are kitted on one pallet and routed on vehicles through the work stations until complete. The components are palletized into kits and finished assemblies removed from the pallets in a kitting station (KS). An assembly vehicle is dedicated to a pallet from the moment when

it picks the pallet up in the KS to the moment when it returns to the KS. After the vehicle drops the finished assembly off, it picks up the next pallet containing another kitted assembly to be built, or if there are no new jobs that require service, remains in the zone adjacent to the KS. The zone is big enough to accommodate all the vehicles, and this is where they park when the system is shut down. Preparation of the kits in a warehouse can also be done by mobile robots visiting the relevant storing area, and abstracted by cyclic processes.

The objective of this work is to present the DES-based framework for the coordination of mobile robots sharing a common 2D motion space, proposed in [12, 15], show its application to the supervisory control of cyclic robot motion processes, and discuss its implementation in a centralized or a distributed controller. The following section describes the control problem for MMRS, gives the assumptions and requirements defining the sought solution. Section 3 explains the assumed discretization scheme of the continuous robot motion space and motion processes. Section 4 describes the RAS (Resource Allocation System) abstraction of concurrent processes [13] and their application to identifying specific classes of MMRS. A model of the feasible dynamic behavior of the robot system is then obtained by mapping the distinguished RAS into DFSA (Deterministic Finite State Automaton) that ensures collision avoidance among the robots. Based on this model, the subsequent section deals with the deadlock avoidance problem, discusses its complexity, and provides relevant algorithms to solve it. Finally, Sect. 6 concentrates on the control architecture implementing the developed control logic, and the last section concludes this research.

## 2 Problem Statement

We consider a Multiple Mobile Robot System (MMRS) viewed as a group of autonomous mobile robots sharing a 2D space. Each robot performs a mission that requires it to travel multiple times along a specific closed path. The path of each robot is planned independently, without taking into account any positional constraints introduced by the paths of other robots. The robots operate asynchronously and are able to control their motion with path-following algorithms that allow each of them to correctly perform its mission when alone on the stage. When sharing the motion space, the robots must refine their motion strategies in order to avoid collisions, through modification of their paths, velocity profiles or both.

The objective of the MMRS control is to ensure that the operation of the system is *correct* and *efficient*. The notion of correctness relates to a qualitative criterion and requires that each robot be able to perform its mission without colliding with other robots. That is, depending on the state of other robots, the path of a robot may be re-planned and/or the robot may have to slow down or even come to a stop and wait until the situation changes and it can safely resume further travel. Thus, the correct control must also ensure for each robot, the possibility to resume its travel after a break, i.e., eliminate from the MMRS behavior such phenomena as deadlocks

and robot starvation. The induced modifications of the robot trajectories inevitably cause an increase of their mission completion time, thus impact MMRS performance measures, whose values can vary depending on the employed conflict resolution policies. Consequently, there are two main questions driving the development of the MMRS control.

1. How to modify dynamically the initially assumed motion control of the robots so that:
  - a. in a finite time interval, all the robots will have accomplished their missions,
  - b. at each moment of this time interval, the areas occupied by any given pair of robots are disjoint.
2. How to induce, within the admissible (i.e. observing requirements (1.a) and (1.b)) robot concurrent operation, efficient MMRS behavior.

As can be noticed, the control satisfying requirements (1.b) and (1.a) ensures, respectively, collision-free and deadlock-free (free of both physical and logical deadlocks) concurrent motion of the robots. Requirement (2) implies the need of a flexible model of MMRS control that leaves room for the optimization of system efficiency and of tools to carry it out. To achieve realization of these postulates, we employ a modular control system, whose subsequent synthesis steps will be discussed in the sequel.

### 3 Discrete Representation of MMRS

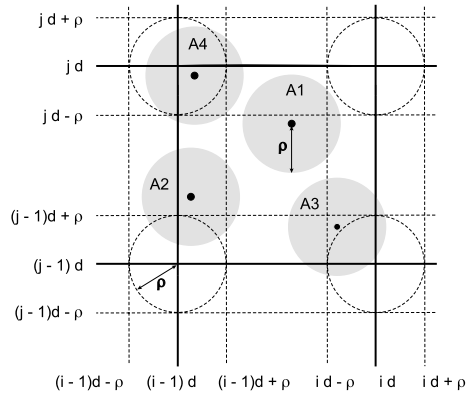
We start from a continuous representation of MMRS. The system consists of a set of mobile robots  $A = \{A_1, \dots, A_n\}$  that share a finite planar workspace  $WS \subset \mathcal{R}^2$  with the  $XY$  coordinate system. Each robot  $A_i \in A$  is represented by a disk with radius  $a_i$ , and its path  $p_i$  is viewed as a curve in  $WS$  that is given by a pair of functions  $p_i(l) = (x_i(l), y_i(l)) \in \mathcal{R}^2$ ,  $l \in [0, \bar{l}_i] \subset \mathcal{R}$ .

The tessellation of the robot motion space that leads to a discrete abstraction of their motion can take place in many different ways. Here, similar to [12], we assume a simple tessellation scheme provided by a grid of horizontal and vertical lines spaced at a distance  $d \geq 2\rho$  and centered at the origin of a coordinate system,  $(x, y)$ , that is superimposed on the motion plane. The resulting cells will be denoted by  $W = \{w[i, j] : i \in \{-\underline{I}, \dots, -1, 0, 1, \dots, \bar{I}\}, j \in \{-\underline{J}, \dots, -1, 0, 1, \dots, \bar{J}\}\}$ , where  $-\underline{I}$ ,  $\bar{I}$ ,  $-\underline{J}$ , and  $\bar{J}$  are taken large enough to encompass the entire area  $\mathcal{U}$ , that supports the robot motion. Then, given a point  $(x, y) \in \mathcal{U}$  and a cell  $w[i, j]$ , we define:

$$(x, y) \in w[i, j] \iff (i - 1) \cdot d \leq x \leq i \cdot d \wedge (j - 1) \cdot d \leq y \leq j \cdot d$$

The size  $d$  of the grid, that defines the length of the cell edges, should be selected by considering the efficiency criteria mentioned above. In general, a smaller value of  $d$

**Fig. 1** Motion space partition (solid line) and regions of constant cell occupation (dashed line)

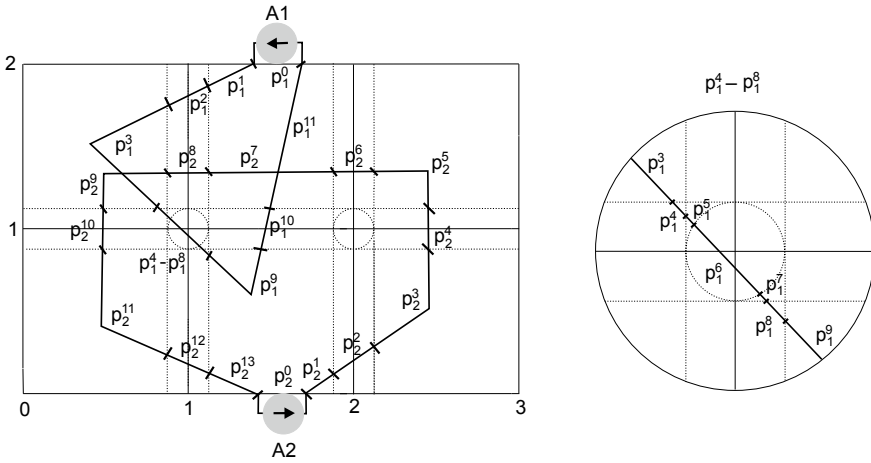


can accommodate a larger number of robots, and therefore, can lead to a higher space occupancy, but at the same time, it will lead to more disruption of the robot travels by the superimposed resource allocation process, and possibly to more congested traffic and longer delays.

In the sequel, we shall say that a robot (with its disk) centered at  $(x^c, y^c)$  occupies cell  $w[i, j]$  iff its disk overlaps the cell, i.e., there exists  $(x, y) \in w[i, j]$  with  $\|(x, y) - (x^c, y^c)\| \leq \rho$ , where  $\|\cdot\|$  denotes the Euclidean norm. A graphical illustration of this concept is given in Fig. 1. More specifically, the adopted tessellation is defined by the grid of the solid horizontal and vertical lines, and the mobile robots are depicted by the grey disks in it. As can be noticed, a robot can occupy one cell (as in the case of A1), two neighboring cells (as in the case of A2), three neighboring cells (as in the case of A3), or four neighboring cells (as in the case of A4).

Moreover, for the considered tessellation scheme, the subset of cells occupied by a mobile robot that is located at  $(x^c, y^c)$  is effectively determined by the relative positioning of  $(x^c, y^c)$  with respect to another partitioning of the motion plane, that is induced by the original tessellation scheme and the robot geometry. In Fig. 1, this induced partitioning is depicted by the dashed lines. If the disk center of a robot is located in one of the circles then the robot occupies all four adjacent cells (as in the case of A4). a robot occupies three cells if it is centered in any region that is the difference between any circle and the square that describes it (as in the case of A3). Next, a robot occupies two cells if its center lies in a rectangle located along the tessellation line (as in the case of A2). And finally, a robot occupies one cell if it is centered in the square located in the middle of the cell (as in the case of A1).

The above characterized tessellation, depicted in Fig. 1 by the dashed lines, partitions robot paths into maximal segments of constant cell occupation. That is, the subset of cells occupied by the robot centered at any of point of a given segment is the same, and it is different from the set of cells occupied by the robot in the sectors preceding and succeeding the considered one. Consequently, it is convenient to view the set of cells  $W$  as the set of MMRS resources, and abstract the motion process of a robot as a sequence of stages, each of which requires for its execution a specific sub-



**Fig. 2** Example paths of two mobile robots and the corresponding resource allocation profiles that are defined by the path partitioning into maximal segments with the same cell occupation. The right part of the figure details the profile obtained for robot A1

**Table 1** The resource allocation induced by the path segmentation of Fig. 2

(a) Robot 1		(b) Robot 2	
Stage No.	Required resources	Stage No.	Required resources
0	$\emptyset$	0	$\emptyset$
1	$w[1, 1]$	1	$w[1, 0]$
2	$w[0, 1], w[1, 1]$	2	$w[1, 0], w[2, 0]$
3	$w[0, 1]$	3	$w[2, 0]$
4	$w[0, 0], w[0, 1]$	4	$w[2, 0], w[2, 1]$
5	$w[0, 0], w[0, 1], w[1, 1]$	5	$w[2, 1]$
6	$w[0, 0], w[0, 1], w[1, 0], w[1, 1]$	6	$w[1, 1], w[2, 1]$
7	$w[0, 0], w[1, 0], w[1, 1]$	7	$w[1, 1]$
8	$w[0, 0], w[1, 0]$	8	$w[0, 1], w[1, 1]$
9	$w[1, 0]$	9	$w[0, 1]$
10	$w[1, 0], w[1, 1]$	10	$w[0, 1], w[0, 0]$
11	$w[1, 1]$	11	$w[0, 0]$
		12	$w[0, 0], w[1, 0]$
		13	$w[1, 0]$

set of resources  $W' \subset W$ . An example partitioning of two paths is demonstrated in Fig. 2. Path  $p_1$  of robot A1 consists of twelve (maximal) segments  $p_1^0-p_1^{11}$ , and path  $p_2$  of robot A2 consists of fourteen such segments,  $p_2^0-p_2^{13}$ . Also, Table 1 specifies the cells occupied by the two robots at the various stages of their route.

## 4 Collision Avoidance in MMRS

The discrete representation of the motion space and robot paths, discussed in the previous section, makes it possible to view MMRS as a sub-class of Resource Allocation Systems (RAS) [13], called FREE-RANGE-RAS [12] and defined as follows.

**Definition 5.1** A *FREE-RANGE-RAS* is defined as a 4-tuple  $\Phi = (\mathcal{R}, C, \mathcal{P}, D)$  such that:

1.  $\mathcal{R} = \{R_1, \dots, R_m\} = W$  is the set of system resources, representing the set of cells.
2.  $C : \mathcal{R} \rightarrow \mathbb{Z}^+$  is the resource capacity function that defines the maximal number of robots that can occupy each particular cell at a time.
3.  $\mathcal{P} = \{P_1, \dots, P_n\}$  is the set of processes, representing the motion of each particular robot  $A_i$  along its path. Each process  $P_i$  is characterized by an ordered set of stages  $\mathcal{E}_i = \{\mathcal{E}_{i1}, \dots, \mathcal{E}_{i,l(i)}\}$  corresponding to the motion of robot  $A_i$  (observed through its disk center) along the consecutive segments of its path.
4.  $D : \bigcup_{i=1}^n \mathcal{E}_i \rightarrow 2^{\mathcal{R}}$  is the resource requirement function that defines the resources  $D(\mathcal{E}_{ij}) = D_{ij}$  required by every process  $P_i$  to execute its each particular stage  $\mathcal{E}_{i,j}$ .
5. The sets of stages  $\mathcal{E}_i$ ,  $i = 1, \dots, n$ , and function  $D$  arise from a geometrical system. That is, there exists a set of planar paths  $p$ , which can be divided into segments  $p_{ij}$ ,  $j = 1, \dots, l(i)$ , traversing the cells so that they induce function  $D$ .

Moreover, we will distinguish two sub-classes of FREE-RANGE-RAS, namely FREE-RANGE- $k$ -RAS, where  $k \in \mathbb{Z}^+$ , and FREE-RANGE\*-RAS.

**Definition 5.2** A *FREE-RANGE- $k$ -RAS* is a *FREE-RANGE-RAS* in which the capacity of each cell  $R \in \mathcal{R}$  is  $C(R) = k$ .

**Definition 5.3** A *FREE-RANGE\*-RAS* is a *FREE-RANGE-RAS* in which for all  $i = 1, \dots, n$ ,  $j = 1, \dots, l(i)$ ,  $|D_{ij}| \in \{1, 2\}$ . That is, no robot ever occupies more than two cells at a time, which is equivalent to that no robot's path overlaps any corner square of the tessellation grid.

A 4-tuple  $\Phi = (\mathcal{R}, C, \mathcal{P}, D)$ , specifies the parameters of a particular MMRS and gives its static abstraction. A dynamic model of MMRS can be developed using the formalism of the Deterministic Finite State Automaton (DFSA) [2], defined as follows.

**Definition 5.4** A deterministic finite state automaton (DFSA) is a 6-tuple  $G = (S, E, \Gamma, f, s_0, S_M)$ , where:

- $S$  is the (finite) set of *states*.
- $E$  is the (finite) set of *events*.
- $\Gamma : S \rightarrow 2^E$  is the *active-event function*. Event  $e \in E$  can occur in state  $s \in S$  iff  $e \in \Gamma(s)$ .

- $f : S \times E \rightarrow S$  is the (partial) *transition function*, defined for pairs  $(s, e)$  such that  $e \in \Gamma(s)$ .  $s' = f(s, e)$  returns the state that results from the occurrence of event  $e$  in state  $s$ .
- $s_0 \in S$  is the *initial state* of  $G$
- $S_M \subseteq S$  is the set of *marked states*

The above DFSA starts its operation from state  $s_0$ . In each state  $s \in S$ , an event  $e$  can only occur if the state transition function  $f()$  is defined for the pair  $(s, e)$ , i.e., if  $e \in \Gamma(s)$ . In that case, we say that event  $e$  is *enabled* in state  $s$ . The occurrence of event  $e$  in  $s$  results in a new state  $s' = f(s, e)$ , which can be changed subsequently by the occurrence of event  $e'$  that is enabled in state  $s'$ , and so on. In order to capture state transitions arising from strings of events, the state transition function  $f$  can be inductively extended to  $S \times E^*$  by the following assumptions:

$$\begin{aligned} \forall s \in S & \quad (f(s, \varepsilon) \equiv s) \\ \forall s \in S \forall u \in E^* \forall e \in E & \quad (f(s, ue) \equiv f(f(s, u), e)) \end{aligned}$$

In the above equations,  $\varepsilon$  denotes the empty string, and  $E^*$  denotes the set of all strings that can be constructed with the elements of the set  $E \cup \{\varepsilon\}$ . Moreover, it is implicitly assumed that the involved single-step transitions correspond to the enabled events, i.e., to the state-event pairs for which the original function  $f$  is defined; otherwise, the extended version of  $f$  is undefined on the corresponding state-string pair. Furthermore, we say that state  $s \in S$  is *reachable* from state  $s_0$  if there exists string  $u \in E^*$  such that function  $f(s, u)$  is defined; the set of all states reachable from  $s$  is called the *reachability set* of  $s$  and denoted by  $Re(s)$ . A special case of such sets,  $Re(s_0)$ , is called the reachability set of the DFSA  $G$ . Using the discussed formalism, a dynamic model of a particular MMRS can be obtained by the following mapping of its specification  $\Phi$  into DFSA.

**Definition 5.5** The DFSA  $G(\Phi) = (S, E, \Gamma, f, s_0, S_M)$  abstracting the feasible dynamics of a FREE-RANGE-RAS  $\Phi = (\mathcal{R}, C, \mathcal{P}, D)$  is defined as follows:

1. The *state set*  $S$  consists of all vectors  $s = (s_1, s_2, \dots, s_n) \in \mathbb{Z}^n$  such that:

$$\begin{aligned} a. \quad & \forall i \in \{1, \dots, n\} & (0 \leq s_i \leq l(i)), \\ b. \quad & \forall R \in \mathcal{R} & (a(s, R) = |\{s_i : R \in D_{i,s_i}\}| \leq C(R)). \end{aligned}$$

Each component  $s_i$  of  $s$  indicates the current stage of process  $P_i$  (the motion process of robot  $A_i$ ). In particular,  $s_i \neq 0$  indicates that robot  $A_i$  is in the  $s_i$ -th path segment of its route, and  $s_i = 0$  indicates that robot  $A_i$  is located off the shared space (thus holding no resource  $R \in \mathcal{R}$ ), where it ends one cycle of its travel and starts another. For each  $R \in \mathcal{R}$ ,  $a(s, R)$  indicates the number of units of resource  $R$  that are allocated in state  $s$ .

2. The *event set*  $E = \{e_i : i = 1, \dots, n\}$ , where for every  $i = 1, \dots, n$ ,  $e_i$  represents the transition of robot  $A_i$  to its next stage.



3. For each pair  $(s, e_i)$ , the *state transition function* returns the new state  $s' = f(s, e_i)$ , whose components  $s'_k, k = 1, \dots, n$ , are given by:

$$s'_k = \begin{cases} (s_k + 1) \bmod (l(k) + 1) & \text{if } k = i \\ s_k & \text{otherwise} \end{cases}$$

4. Function  $f()$  is defined for a pair  $(s, e)$  iff  $e \in \Gamma(s)$ , where the *set of feasible events*  $\Gamma$  is defined by  $\Gamma(s) \equiv \{e \in E \mid s' = f(s, e) \in S\}$ .
5. The *initial state*  $s_0 = \mathbf{0}$ , which corresponds to the situation where all the robots are in their private space and therefore, all the system resources are free.
6. The *set of marked states*  $S_M$  is the singleton  $S_M = \{s_0\}$ , and it expresses the requirement for complete process runs.

If  $\Phi \in \text{FREE-RANGE-1-RAS}$ , that is, if the capacity of all cells is  $C(R_i) = 1$  then the above defined DFSA model enforces mutually exclusive occupation of the cells by the robots and, consequently, their collision-free motion. Otherwise, that is, if for some cell  $R_i, C(R_i) > 1$  then still no collisions can occur among robots that have been allocated disjoint sets of cells, but an additional local coordination system is needed to prevent internal collisions of the robots within each cell  $R_i$ . Such a system can be based on a DES model, obtained by further discretization of the cell, creating a local FREE-RANGE-1-RAS, or employ some of the reactive collision avoidance methods, e.g., based on the potential field [9].

## 5 Deadlock Avoidance in MMRS

The operation of the *MMRS* model  $G(\Phi)$ , ensuring the disjoint motion of the robots' disks, satisfies Requirement (1.b) defined in Sect. 2. However, Requirement (1.a) is not satisfied, as the reachability set of  $G(\Phi)$  can contain states  $s$  that are not *safe*, i.e., such that the initial state  $s_0$  is not reachable from  $s$ .

To observe this, consider again the example of two robots and their path segmentation, depicted in Fig. 2, and the resource requirements induced by this path segmentation, given in Table 1. Let us assume that  $\Phi \in \text{FREE-RANGE-1-RAS}$ , that is, the capacity of the cells is 1. Next notice that in the initial state event sequence  $u = e_1, e_2, e_2, e_1, e_2, e_2, e_1, e_2$  is feasible and drives the system from state  $s_0 = [0, 0]$  to state  $s_1 = f(s_0, u) = [3, 5]$ . Then two state transitions are feasible: to state  $s_2 = f(s_1, e_1) = [4, 5]$  and to state  $s_3 = f(s_1, e_2) = [3, 6]$ . State  $s_2 = [4, 5]$  is safe, as the initial state  $s_0$  can be reached from it by completing first the cycle of robot  $A_1$  and then completing the cycle of robot  $A_2$ . In state  $s_3 = [3, 6]$  only two event sequences are feasible:  $u' = e_1, e_2$  and  $u'' = e_2, e_1$ , and both drive the system to state  $s_4 = f(s_3, u') = f(s_3, u'') = [4, 7]$ , which is a deadlock as  $\Gamma(s_4) = \emptyset$ . No event is feasible in  $s_4$  because for its next stage, robot  $A_1$  requires resource  $w[1, 1]$ , which is held by robot  $A_2$ , and robot  $A_2$  requires resource  $w[0, 1]$ , which is held by robot  $A_1$ .

In order to enforce the correct operation of  $G(\Phi)$ , it is necessary to introduce a *supervisor* that extends the *feasible-event function*  $\Gamma(s)$  to a more restrictive *admissible-event function*. The supervisor disables the occurrence of some state transitions and thus constrains the behavior of MMRS so that for each admissible event sequence, there exists its admissible extension driving the system to the initial state. This makes the system *reversible*, hence deadlock-free, and allows each process to repeat its cycle any arbitrary number of times.

The optimal, i.e., the least restrictive supervisor should accept an event  $e \in E$  in state  $s \in S$  if and only if  $e \in \Gamma(s)$  (event  $e$  is enabled in state  $s$ ) and the next state  $s' = f(s, e)$  is safe. Thus, any algorithm to check these conditions must solve the following problem.

**Safety problem:** Given a FREE-RANGE-RAS  $\Phi = (\mathcal{R}, C, \mathcal{P}, D)$ , a safe state  $s \in R(s_0)$  and an enabled event  $e \in \Gamma(s)$ , find out whether or not state  $s' = f(s, e)$  is safe.

As demonstrated in [11], the Safety problem is NP-complete even if addressed to any of the sub-classes of RAS, FREE-RANGE- $k$ -RAS,  $k \in \mathbb{Z}^+$ . On the other hand, there exists a polynomial algorithm solving the safety problem for systems  $\Phi \in \text{FREE-RANGE}^*\text{-RAS}$ , in which the capacity of each resource  $R \in \mathcal{R}$  is  $C(R) > 1$  [15]. The high complexity of the first group of problems implies that practically only sub-optimal solutions of the safety problem can be considered in FREE-RANGE-RAS. Such algorithms ensure the reachability of the initial state, but not necessarily in the least restrictive way.

From the viewpoint of the control synthesis for MMRS, most useful appear two of the sub-classes of FREE-RANGE-RAS, each of which has its pros and cons. The first class, FREE-RANGE-1-RAS, assumes that no more than one robot at a time can be present in each particular cell, thus no further local coordination is required. However, the supervisory control employs a sub-optimal algorithm, which is in general overly restrictive and may have some negative impact on the efficiency of the system. The second class, FREE-RANGE\*-2-RAS, allows for a maximally permissive supervisor, so no unnecessary event disabling ever happens. Yet, it imposes some constraints on the shape of the paths, which should omit the corner squares of the tessellation structure. Also, as more than one robot can be present in a cell at a time, an additional control is needed to ensure then their collision-free motion. Such a coordination is, however, fairly simple, as the maximal number of robots in a cell is limited to two. In the following we present two supervisors for the two distinguished models, as proposed in [12, 15], respectively.

## 5.1 Deadlock Avoidance in FREE-RANGE-1-RAS

In these systems, the reversibility of  $G'(\Phi)$  can be enforced by constraining the reachability space  $Re(s_0)$  of  $G(\Phi)$  to the subspace of *p-ordered* states  $Re'(s_0) \subseteq Re(s_0)$ . The definition of such states, given below, uses the notion of a *private stage*

of process  $P_i$ ,  $\mathcal{E}_{iq}$ , which means that the resources required at this stage,  $D_{i,q}$ , are not required by any other process in order to complete their cycle and reach back their initial state.

**Definition 5.6** In system  $G(\Phi)$ , state  $s = (s_1, \dots, s_n)$  is *p-ordered* iff there exists an order on the set of robots  $\mathcal{A}$ ,  $p : \mathcal{A} \rightarrow \{1, 2, \dots, n\}$  that satisfies the following condition:  $\forall i, j$  s.t.  $p(A_j) > p(A_i)$ ,  $\forall k = s_i..q_i$ ,  $D_{ik} \cap D_{js_j} = \emptyset$ , where  $q_i$  is the smallest number s.t.  $q_i \geq s_i$  and stage  $\mathcal{E}_{iq_i}$  is private, if such a number exists. Otherwise  $q_i = 0$ .

Less formally, a state  $s$  is p-ordered iff there exists an order of the robots such that no robot with higher order occupies any of the cells that lie on the way of a robot with a lower order to its nearest private stage. A procedure checking whether or not this property is observed by a particular state of a particular system  $G(\Phi)$  is presented in Algorithm 1.

---

**Algorithm 1.** The function testing the p-ordered property of states in FREE-RANGE-1-RAS

---

**Input** : Parameter  $\Phi$  describing the RAS, state  $s \in S$ .

**Output**: *True* if the state is p-ordered, *false* otherwise.

```

1 Function p-ordered( $\Phi, s$ ) : bool
2    $\mathcal{A} \leftarrow \mathcal{P}$ ;
3   occupied  $\leftarrow \emptyset$ ;
4   for  $i = 1, \dots, n$  do
5     occupied  $\leftarrow$  occupied  $\cup D_{is_i}$ ;
6      $q_i \leftarrow \text{minpriv}(i, s_i)$ ; remain[ $A_i$ ]  $\leftarrow \bigcup_{j=s_i}^{q_i} D_{i,j}$ ;
7   repeat
8      $\mathcal{A}' \leftarrow \mathcal{A}$ ;
9     for  $A_i \in \mathcal{A}$  do
10      if remain[ $A_i$ ]  $\cap$  occupied =  $D_{is_i}$  then
11         $\mathcal{A} \leftarrow \mathcal{A} \setminus \{A_i\}$ ;
12        occupied  $\leftarrow$  occupied  $\setminus D_{i,s_i}$ ;
13      if  $\mathcal{A} = \emptyset$  then
14        p-ordered  $\leftarrow$  TRUE;
15      else
16        p-ordered  $\leftarrow$  FALSE;
17   until  $\mathcal{A}' = \mathcal{A} \vee \mathcal{A} = \emptyset$ ;

```

---

As in the above algorithm, the operations on the set  $\mathcal{A}$  are  $O(n)$  complex, the complexity of the whole function is  $O(n^2)$ , which qualifies it for online applications. It is also not hard to notice the following property.

**Property 5.1** *In system  $G(\Phi)$ , the final state can be reached from any state  $s$  that is p-ordered.*

*Proof* The condition defining the p-ordered state provides the robots  $A_i$ ,  $i = 1..n$ , with the ability to progress one-by-one, in the order given by  $p(A_i)$ , to their respective closest private stages  $\mathcal{E}_i^{q_i}$ . Since then no robot occupies a cell that can be required by any other robot on its way to complete the cycle, the robots can one by one reach back their initial state. ■

**Definition 5.7** A p-controlled MMRS  $G'(\phi)$  is a restriction of DFSA  $G(\Phi) = (S, E, \Gamma, f, s_0, S_M)$  obtained by:

- substituting the *feasible-event function*  $\Gamma(s)$  by *admissible-event function*  $\Gamma'(s) = \{e : e \in \Gamma(s) \wedge s' = f(s, e) \text{ is p-ordered}\}$ , and
- substituting the transition function  $f$  with  $f'$  such that  $f'(s, e) = f(s, e)$ , but it is only defined for admissible pairs  $(s, e)$ , i.e., such that  $e \in \Gamma'(s)$ .

The following theorem proves that  $G'(\Phi)$  is reversible and thus its operation satisfies Requirement (1.a).

**Theorem 5.1** *In a p-controlled MMRS  $G(\Phi)$ , the initial state  $s_0$  is reachable from each state  $s$  reachable from the initial state  $s_0$ .*

*Proof* Based on Definition 5.7, each state reachable in a p-controlled  $G(\Phi)$  is p-ordered. Thus, by Property 5.1, the theorem holds. ■

## 5.2 Deadlock Avoidance in FREE-RANGE\*-2-RAS

Since in this class of models the robot paths omit the square corners of the tessellation grid, the motion process of each robot consists of a sequence of stages that correspond alternately to the travel in a cell and the transition from one cell to the next one. At a stage of the latter type, a robot occupies both cells, yet it eventually passes to a stage of the former type and its disk no more occupies the previous cell. Thus, from the viewpoint of deadlock avoidance, the transitions between the cells can be considered as transient states, and FREE-RANGE\*-2-RAS can be viewed as a system of processes, whose each stage  $\mathcal{E}_{ij}$  requires a single resource  $D(X_{ij}) = D_{ij} \in \mathcal{R}$ .

The supervisor defined for this class of MMRS control models employs a graphical representation of a state that has the form of *resource allocation graph*.

**Definition 5.8** The resource allocation graph representing a state  $s \in S$  of  $G(\Phi)$  is a graph  $F(s) = (V, H)$  such that:

- The set of vertices is defined by the extended set of resources  $V = \mathcal{R} \cup \{R_\infty\}$ , where  $R_\infty$  is a dummy resource of infinite capacity allocated to each process  $P_i \in \mathcal{P}$  at its stage  $\mathcal{E}_{i0}$ .

---

**Algorithm 2.** The function testing the safety of state transition  $s' = f(s, e_i)$  in FREE-RANGE\*-2-RAS.

---

**Input:** Parameter  $\Phi$  describing the RAS, state  $s$ , and the index  $i$  of the process, whose potential advancement to the next stage is considered in the context of the safety of the resulting state  $s'$ .

```

1 Function safe( $\Phi, s, i$ ) : bool
2   if  $s_i = l(i)$  then
3     return true
4   if  $a(s, D_{i s_{i+1}}) = |\{s_k : D_{k, s_k} = D_{i s_{i+1}}\}| = 0$  then
5     return true
6    $s' \leftarrow f(s, e_i)$ ;
7   if  $\exists t = R^1, R^2, \dots, R^q, q \geq 1$ , such that  $R^1 = D_{i s_{i+1}}$  and  $R^q = R_\infty$  or
    $a(s', R^q) = |\{s'_k : D_{k, s'_k} = R^q\}| < 2$  then
8     return true
9   return false

```

---

- The set of edges is defined by the set of robot processes  $H = \mathcal{P}$ . The edge (corresponding to process)  $P_i$  goes from vertex  $R \in \mathcal{R}$  to vertex  $R' \in \mathcal{R}$  iff, at state  $s$ , process  $P_i$  has been allocated resource  $R$  and for its next stage it requires resource  $R'$ . Edge  $P_i$  goes from vertex  $R \in \mathcal{R}$  to vertex  $R_\infty$  iff  $s_i = l(i)$ , i.e., at state  $s$ , process  $P_i$  executes the last stage of its cycle,  $\mathcal{E}_{i, l(i)}$ . Edge  $P_i$  goes from vertex  $R_\infty$  to a vertex  $R \in \mathcal{R}$  iff  $s_i = 0$ , that is, process  $P_i$  is in its initial state.

The following theorem [14] provides a property that allows the construction of a maximally permissive supervisor.

**Theorem 5.2** Consider a RAS  $\Phi \in \text{FREE-RANGE}^* \text{-2-RAS}$ , a DFSA  $G(\Phi)$ , a reachable safe state  $s$ , and an event  $e_i$  such that the next state  $s' = f(s, e_i)$  is defined. Then, state  $s'$  is safe iff in the graph  $F(s')$ , there exists a path  $t = R^1, R^2, \dots, R^q$ ,  $q \geq 1$ , from resource  $R^1 = D_{i s_{i+1}}$ , to a resource  $R^q \in V$  that in state  $s'$  is allocated to fewer processes than its capacity.

It is clear that the restriction of any DFSA  $G(\Phi)$ ,  $\Phi \in \text{FREE-RANGE}^* \text{-2-RAS}$ , to  $G'(\Phi)$  obtained by substituting function  $\Gamma(s)$  with function  $\Gamma'(s) = \{e : e \in \Gamma(s) \wedge s' = f(s, e) \text{ is safe}\}$  yields a model of MMRS that is reversible and maximally permissive, i.e., it captures all the trajectories of  $G(\Phi)$  that reach the initial state and no state  $s \in Re(s_0)$  from which the initial state is not reachable.

When verifying the safety condition, there is no need to construct graph  $F(s')$ , from scratch at each state change  $s' = f(s, e)$ , as it can be directly obtained by a small update of  $F(s)$ —removing one edge and adding another. Moreover, it is possible to distinguish two special cases of state  $s$  when the safety condition holds: (i)  $s_i = l(i)$ , as then  $D_{i s_{i+1}} = R_\infty$  and its capacity is infinite, and (ii) resource  $R^1 = D_{i s_{i+1}}$  is not allocated to any process in state  $s$ . Thus, checking the safety of a state transition can be done with the function specified in Algorithm 2. The most complex part of the

calculations is testing the existence of the required path in graph  $F(s')$ , which can be done with, e.g., the depth first search, that has the  $O(|V| + |H|)$  computational complexity.

## 6 Implementation of the MMRS Control Logic

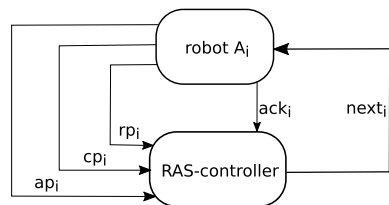
The logic described in the previous sections can be implemented both in a centralized and distributed manner. In the former case, all the robots communicate with the central RAS controller. In the latter case, each robot is equipped with a local RAS-controller, a higher control level that interacts with its lower control level in the same way as in the centralized case, but, additionally, it has to communicate with other robots in order to be aware of their state. The specifics of the distributed implementation of the FREE-RANGE-RAS based control can be found in [15], whereas here we focus on the common features of both approaches.

### 6.1 Interaction of Robots with Their RAS-Controller

The interaction of the robots and the RAS-controller is event-based, as depicted in Fig. 3. The controller generates only one type of events,  $next_i$ ,  $i = 1, \dots, n$ , which is a permission for robot  $A_i$  to proceed to the next path segment. Having received this message, the robot confirms it with the  $ack_i$  signal. Next, each robot  $A_i$  informs the controller about the occurrence of three types of events,  $ap_i$ ,  $cp_i$ , and  $rp_i$ , corresponding to reaching three types of characteristic points on its path:  $ap$ ,  $cp$ , and  $rp$ .

- Event  $ap_i$  is generated when robot  $A_i$  passes an *approach point*  $ap$ , which signals that  $A_i$  is approaching its next stage. These points are distinguished at the end of the path segments  $p_{ij}$  such that the transition to the next path segment  $p_{i,j+1}$  requires allocation of additional resources  $D_{i,j+1} \setminus D_{ij} \neq \emptyset$ .
- A *critical point*  $cp$  is set at a safe-braking distance from the end of each path segment  $p_{ij}$  in which appears point  $ap$ . If by arriving at point  $cp$ , robot  $A_i$  has not received the signal  $next_i$ , it generates event  $cp_i$  and starts decelerating. At the

**Fig. 3** Interaction of the RAS-controller with a robot



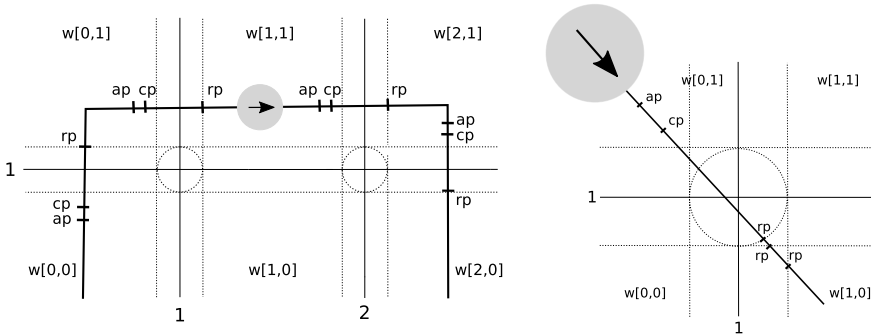


Fig. 4 Two exemplary paths with their characteristic points

arrival of signal  $next_i$ , which can occur after the robot has come to the standstill or is still decelerating, it accelerates again and proceeds to the next section.

- A *release point*  $rp$  is set at the border of two path segments  $p_{ij}$  and  $p_{i,j+1}$  such that  $D_{i,j+1} \subset D_{ij}$ . On passing a point  $rp$ , robot  $A_i$  generates event  $rp_i$  to inform the controller that it can deallocate from the robot the resources  $D_{ij} \setminus D_{i,j+1}$ .

An example of a robot path with its characteristic points is given in Fig. 4. If a path segment  $p_{ij}$  is very short, in particular if it is equal or shorter than the safe-braking distance, then such a segment can be merged with the next segment  $p_{i,j+1}$  or the previous stage  $p_{i,j-1}$ , creating a process stage that requires the union of the resources required by both stages,  $D_{ij} \cup D_{i,j+1}$  or  $D_{i,j-1} \cup D_{ij}$ , respectively. In the right part of Fig. 4, three path segments preceding the robot were merged: the segment between the dashed horizontal and vertical line, the segment between the vertical line and the ‘square’, and the segment between the ‘square’ and the ‘circle’. The resulting path segment starts in cell  $w[0, 1]$  at the cross of the path with the dashed horizontal line, ends in cell  $w[0, 1]$  at the cross of the path with the ‘circle’, and its resource requirement is  $\{w[0, 0], w[0, 1], w[0, 1], w[1, 1]\}$ . All these cells should be allocated before the robot crosses the horizontal line, so one pair of points  $ap$  and  $cp$  is sufficient. There is no need, however, to merge the symmetric path segments in cell  $w[0, 1]$ , between the ‘circle’ and the ‘square’, and between the horizontal and vertical lines, as the resources are only released here. Since no new resource allocation is needed, there is no risk that the robot will have to brake, thus there is no lower limit on the path segment length.

## 6.2 Priority Control

The decisions made by the RAS controller concern the selection of events that will be activated in a particular state through signals  $next_i$ . To make such decisions, the controller must follow the state of the robots, updating it at the occurrence of

each event that is sent/received to/from any robot. Then it checks the admissibility conditions established in Sects. 4, 5, and solves the conflicts among the robots using some *priority policy* in order to decide which robots should be allowed to continue their motion, and which should be temporarily suspended in the case when concurrent operation of all of them is not admissible. By a priority policy we understand an algorithm that, based on some heuristic priority criterion, selects a subset of events  $\Delta(s) \subseteq \Gamma'(s)$  such that:

- (i) Each pair of events  $e_i, e_k \in \Delta(s)$  is non-conflict, that is, occurrence of one of them does not make inadmissible the other. Hence, robots  $A_i$  and  $A_k$  can transit to their next sectors concurrently.
- (ii) For each event  $e_i \notin \Delta(s)$ , there exists event  $e_k \in \Delta(s)$  that is in conflict with  $e_i$ , i.e., granting robot  $A_k$  the permission to transit to its next sector makes the transition of robot  $A_i$  inadmissible.

The set  $\Delta(s)$  is recalculated at each state change, and command *next* is sent to all robots  $A_i$  that have passed their approaching point *ap* and  $e_i \in \Delta(s)$ . The algorithms to calculate  $\Delta(s)$  can range from simple priority rules to optimization algorithms based on predicted time parameters of the robot motion, yet, since executed online, they must feature low computational complexity. Since the proposed RAS controller can serve both a real MMRS and its simulator, the selection of the most efficient policy, with respect to an assumed criterion, can be done experimentally.

## 7 Remarks and Conclusions

Although it is possible to view the operation of a group of mobile robots as a set of processes sharing common resources, there is a number of features that seriously differ this system from other systems typically represented by a similar abstraction, e.g., job shops. The motion processes of the robots and their shared resource—the motion space are not inherently discrete, but have a continuous character, and when viewed as a resource sharing system require discretization. The system is deadlock-prone, and the problem of deciding whether or not a particular resource allocation is safe in a particular state is NP-complete. The robots operate asynchronously and their travel time between distinguished points can be only roughly estimated. The actual time can substantially vary from the initially assumed one. This is due to inaccurate time calculation based, e.g., on the robot control model as well as possible on-line modification of its path or velocity profile necessary for collision avoidance with other robots. Moreover, vehicle transport systems, are often subordinate to other systems, e.g., a machining system in a manufacturing plant. That is, the start moments of particular transport operations depend on the activity of the latter system, which makes the prediction of the time behavior of the former system still more complex.

All the above features imply the need of a DES-based robot coordination system that calculates control decisions online, based on the current state of the robots, rather



than execution of an offline calculated schedule. The presented work advocates such an approach.

In this chapter we recaptured our earlier results concerning the supervisory control synthesis for free-ranging mobile robot systems within the framework of the RAS model, showed its application to the supervisory control of cyclic robot motion processes, and discussed implementation of this concept in a RAS-controller. Currently we are focused on the development of a MMRS controller employing the presented theory for a fleet of real mobile robots, which will be followed by experimental research.

**Acknowledgements** This work was partially supported by grant no. 2016/23/B/ST7/01441 of the National Science Center.

## References

1. Barraquand, J., Latombe, J.: Robot motion planning: a distributed representation approach. *Int. J. Robot. Res.* **10**(6), 628–649 (1991)
2. Cassandras, C.G., Lafortune, S.: *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston (1999)
3. Chang, C., Chung, J., Lee, B.: Collision avoidance of two robot manipulators by minimum delay time. *IEEE Trans. Syst., Man, Cybern.* **24**(3), 517–522 (1994)
4. Chang, D., Shadden, S., Marsden, J., Olfati Saber, R.: Collision avoidance for multiple agent systems. In: *Proceedings of 42nd IEEE International Conference on Decision and Control*, Maui, HI, USA, 9–12 December 2003, pp. 539–543. IEEE (2003)
5. Cirillo, M., Pecora, F., Andreasson, H., Uras, T., Koenig, S.: Integrated motion planning and coordination for industrial vehicles. In: *ICAPS'14 Proceedings of the Twenty-Fourth International Conference on International Conference on Automated Planning and Scheduling*, Portsmouth, New Hampshire, USA, 21–26 June 2014, pp. 463–471. AAAI Press (2014)
6. Claes, D., Tuyls, K.: Multi robot collision avoidance in a shared workspace. *Auton. Robot.* **42**(8), 1749–1770 (2018)
7. Draganjac, I., Miklic, D., Kovacic, Z., Vasiljevic, G., Bogdan, S.: Decentralized control of multi-AGV systems in autonomous warehousing applications. *IEEE Trans. Autom. Sci. Eng.* **13**(4), 1433–1447 (2016)
8. Ferrera, E., Capitán, J., Castaño, A.R., Marrón, P.J.: Decentralized safe conflict resolution for multiple robots in dense scenarios. *Robot. Auton. Syst.* **91**, 179–193 (2017)
9. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.* **5**(1), 90–98 (1995)
10. Kousi, N., Koukas, S., Michalos, G., Makris, S.: Scheduling of smart intra-factory material supply operations using mobile robots. *Int. J. Prod. Res.* **57**(3), 801–814 (2018)
11. Reveliotis, S., Roszkowska, E.: On the complexity of maximally permissive deadlock avoidance in multi-vehicle traffic systems. *IEEE Trans. Autom. Control.* **55**(7), 1646–1651 (2010)
12. Reveliotis, S., Roszkowska, E.: Conflict resolution in free-ranging multivehicle systems: a resource allocation paradigm. *IEEE Trans. Robot.* **27**(2), 283–296 (2011)
13. Reveliotis, S.A.: *Real-time Management of Resource Allocation Systems: A Discrete Event Systems Approach*. Springer, New York (2005)
14. Roszkowska, E., Goral, I.: Correct-by-construction distributed control for multi-vehicle transport systems. In: *IEEE International Conference on Automation Science and Engineering*, Madison, Wisconsin, USA, 17–21 August 2013, pp. 156–161. IEEE (2013)

15. Roszkowska, E., Reveliotis, S.: A distributed protocol for motion coordination in free-range vehicular systems. *Automatica* **49**(6), 1639–1653 (2013)
16. Snape, J., van den Berg, J., Guy, S.J., Manocha, D.: The hybrid reciprocal velocity obstacle. *IEEE Trans. Robot.* **27**(4), 696–706 (2011)
17. Valle, S.M.L., Hutchinson, S.A.: Optimal motion planning for multiple robots having independent goals. *IEEE Trans. Robot. Autom.* **14**, 912–925 (1998)

# Blockage-Free Route Planning for In-Plant Milk-Run Material Delivery Systems



Grzegorz Bocewicz , Izabela Nielsen  and Zbigniew Banaszak 

**Abstract** In this chapter, two kinds of intertwined decisions regarding the movement of vehicles in an in-plant milk-run delivery system are considered: routing decisions, which determine the set of sequences of stations visited by each tugger train, and scheduling decisions, which plan congestion free movement of the tugger trains. The problem under study, called the Multi Trip and Multi Cycle Pick-up and Delivery Problem with Time Windows and Congestion Free Traffic, can be viewed as an extension of the pick-up and delivery problem with time windows in which multiple tugger trains travel along closed-loop congestion-free routes in different cycles. A declarative model of the investigated milk-run delivery principle makes it possible to formulate a vehicle routing and scheduling problem the solution to which determines the route, the time schedule, and the type and number of parts that the different trucks must carry to fulfill orders from various customers/recipients. Due to the requirement of congestion-free milk-run traffic, a scheduling period slicing principle allowing to synchronize cyclic flows of different periods is applied. Its implementation, resulting in a cyclic schedule composed of quasi cyclic sub-schedules, implies a recursive formulation of a well-known constraint satisfaction problem. The goal is to find solutions that can minimize both vehicle downtime and the takt time of the production flow. Computer experiments illustrate the possibility of using the present approach in real-life systems.

---

G. Bocewicz (✉) · Z. Banaszak  
Faculty of Electronics and Computer Science, Koszalin University of Technology,  
2 Śniadeckich St., 75-453 Koszalin, Poland  
e-mail: [bocewicz@ie.tu.koszalin.pl](mailto:bocewicz@ie.tu.koszalin.pl)

Z. Banaszak  
e-mail: [zbigniew.banaszak@ie.tu.koszalin.pl](mailto:zbigniew.banaszak@ie.tu.koszalin.pl)

I. Nielsen  
Department of Materials and Production, Aalborg University, Fibigerstræde 16,  
9220 Aalborg Øst, Denmark  
e-mail: [izabela@mp.aau.dk](mailto:izabela@mp.aau.dk)

## 1 Introduction

In this chapter, we consider a new class of problem, dubbed the Multi Trip and Multi Cycle Pick-up and Delivery Problem with Time Windows and Congestion Free Traffic (MTMC-PDPTW-CFT), which generalizes a pick-up and delivery problem with time windows in which multiple tugger trains follow their multiple closed-loop and congestion-free routes in different cycles. In this context, taking into account the limited number of vehicles (tugger trains) and their capacity constraints as well as the time windows constraining congestion free execution of the pick-up and delivery operations, a corresponding milk-run planning problem can be seen as a special case of the well-known hard combinatorial optimization vehicle routing problem [7, 18].

The goal of milk-run planning is to minimize the number of tugger trains required to perform the services, via minimizing cycle time [10]. Of course, in a general case, the problem in which cyclic transport routes linking workstations and/or production cells composed of sets of machines are sought for given production routes, as well as the reverse problem in which production routes are sought and the transport routes are given, are combinatorial NP-complete problems [1, 2, 12]. Also, it should be noted that concurrent operation of multiple vehicles in a limited layout of a distribution network may lead to congestion resulting in deadlocks, livelocks, collisions, overcrowdings and so on [21]. Consequently, since the problem of defining a set of feasible routing solutions allowing conflict-free movement of vehicles through a given layout of a distribution network is an NP-hard decision problem [14], the solutions to a MTMC-PDPTW-CFT of a size encountered in practice have an acceptable but not optimal.

Therefore, in this study we try to find a computationally effective approach aimed at simultaneous routing and scheduling of tugger train flow as well as design of a distribution network infrastructure. For that reason the should allow to formulate a decision problem that captures the importance of striking an equilibrium between potential expectations regarding milk-run traffic and the capacity of the existing distribution network. Moreover it should focus on resolving resource conflicts, i.e. conflicts that arise when different activities simultaneously request access to shared resources (e.g. intersections and/or guideway line segments) of limited quantity.

In the MTMC-PDPTW-CFT setting, a homogeneous fleet of vehicles operates multi-tour routes out of a supermarket and warehouse to deliver and pick-up loads (parts) to and from workstations and/or manufacturing cells (workstations for short). In order to provide timely deliveries, the loads must be brought from the supermarket within the time windows in accordance with the respective workstations requirements. In turn, the different workstations have loads that must be picked-up, within the workstation time windows, and brought to the warehouse. In other words, this study assumes that tugger trains travel cyclically along routes, servicing workstations and terminals, and that the set of routes guarantees that all workstations are serviced according to the given multiproduct production flow schedule. The characteristics of the MTMC-PDPTW-CFT that set it apart from other vehicle routing problems include:

1. multi-commodity demand defined as specific, time-dependent origin-to-destination loads to be delivered or picked-up in a pre-defined sequence of delivery/pick-up time windows;
2. synchronization of tugger train access to shared segments of transportation routes;
3. multi-tour cyclic delivery/pick-up processes running at different periods.

Given these characteristics, we focused on a class of multi-assortment production systems typically used in the automotive industry.

In the production system considered, composed of  $m$  workstations, different products  $n$  are manufactured at the same time. Some of the workstations, which are located along the production routes of various products, are used by them alternately. The production takt time  $TP_i$  of every  $i$ th product (job  $J_i$ ) is determined by the bottleneck of the associated production route, i.e. the slowest workstation along that route. Assuming that workpieces are manufactured at this workstation in batches  $B_i$ , it is easy to notice that in the period determined by the smallest common multiple, i.e.  $T = LCM(B_1 \cdot TP_1, \dots, B_i \cdot TP_i, \dots, B_n \cdot TP_n)$ , a part set  $(\frac{T}{TP_1} \cdot B_1, \dots, \frac{T}{TP_i} \cdot B_i, \dots, \frac{T}{TP_n} \cdot B_n)$  is produced. This means that during the period  $T$ , workpieces are produced along the various routes in batches of various sizes. Moving along the individual production routes, these batches are delivered from one workstation to the next and are then picked up from them in appropriate time windows that make up sequences  $PR_i = (TD_{i,1}, TC_{i,1}, \dots, TD_{i,j}, TC_{i,j}, \dots, TD_{i,r}, TC_{i,r})$ , where  $TD_{i,j}, (TC_{i,j})$  is a time window in which a production batch  $B_i$  is delivered (picked up) to (from) the  $j$ th workstation along the  $i$ th route. It is easy to see that the opening time of the window in which batch  $B_i$  is picked up from the  $j$ th workstation is  $\lceil TC_{i,j} \rceil \geq \lceil TD_{i,j} \rceil + B_i \cdot t_j$ , where:  $\lceil TD_{i,j} \rceil$  – the moment of closing of the delivery window,  $t_j$  – time of technological operation related to the production of one batch of workpieces manufactured on  $j$ th workstation on the  $i$ th route. The time windows in which batches are delivered  $TD_{i,j}, TD_{i,j+1}$  and picked up  $TC_{i,j}, TC_{i,j+1}$  occur at intervals determined by the takt time of the production batch of the  $i$ th workpiece, i.e.  $B_i \cdot TP_i$ .

For a production flow organized in this way, a milk-run system is sought which allows production batches to be distributed in the given time window sequences  $PR_i$  subject to the limitations of the available route layout. In other words, knowing a set of sequences of pick-up and delivery windows for production batches  $PR_i$ , a route layout and the size of available tugger train fleet, one looks for routes and the associated schedules that allow collision-free movement of tugger trains and just-in-time delivery and pick-up of workpieces and products. Of course, one vehicle can make multiple trips per cycle.

To the best of our knowledge, the MTMC-PDPTW-CFT has not been addressed in the literature before. Given this, our goal is to formally introduce its reference model and provide a mathematical formulation of the problem in the conceptual framework of declarative modeling. In other words, we want to find a computationally effective approach that will allow to simultaneously route and schedule tugger train flow and design the infrastructure of a distribution network. Put yet differently, the reference model sought, which allows to formulate a decision problem that captures

the importance of striking an equilibrium between potential expectations regarding milk-run traffic and the capacity of the existing distribution network, focuses on resolving resource conflicts that arise when different delivery/pick-up activities simultaneously request access to the shared time windows.

A long-term objective of the present study is to develop a method, derived from the reference model, oriented toward the use of decision-support-system-like software. With this in mind, we employ the declarative modeling framework, mostly because of its fast prototyping capability. It should be recalled that in declarative models, focus is on what the solution is. In other words, in contrast to imperative models of computation, which are expressed in terms of states and sequences of state-changing operations and take an “inside-out” approach, i.e. simply describe how a solution is obtained, declarative models take an “outside-in” approach. Instead of describing how a process has to work exactly, a declarative model specifies only the essential characteristics of the process. The present study is a continuation of our previous work on methods of fast prototyping of solutions to selected problems of routing, batching and scheduling of tasks typically performed in batch flow production systems, as well as problems regarding the planning and control of production flow in a class of multi-assortment production systems typically used in the automotive industry [3–5].

In what follows, we: (1) formally define and present a formulation of the Multi Trip and Multi Cycle Pick-up and Delivery Problem with Time Windows and Congestion Free Traffic, (2) propose a declarative model to address the problem of MTMC-PDPTW-CFT within the framework of recursive CSP, which allows one to consider processes with different cycles, (3) propose an approach based on the principle of “slicing the scheduling period into time slots”, which allows to synchronize cyclic flows of different periods resulting in a cyclic schedule composed of quasi cyclic schedules, (4) introduce the deadlock prevention conditions allowing one to prototype alternative congestion free milk-run routes and schedules, (5) analyse the performance of milk-run traffic systems with different numbers of trips allowed, executed within a period  $T$ , and (6) analyse the performance of the proposed method and study the impact of two main characteristics of the problem, namely the configuration of admissible congestion free routings, and the number of time slots in a scheduling period.

The remainder of the paper is organized as follows. Section 2 reviews the literature. Section 3 presents a motivating example followed by formulation of the Multi Cycle Pick-up and Delivery Problem with Time Windows and Congestion Free Traffic, and a detailed description of the problem within the framework of recursive CSP. The proposed methodology is described in Sect. 4. Computational results are then reported and analysed in Sect. 5, while conclusions and future work are considered in Sect. 6.

## 2 Related Work

It can be shown that the total cost spent in a milk-run delivery process is lower than that incurred by applying the direct shipment method [23]. This means that regular pick-up/delivery of workpieces by the milk run method is more effective than the use of the direct or the collaborative transportation methods. Typically, milk-run “trains” consisting of a tugger and three to five trailers use fixed routes. Tugger trains can be shared by multiple suppliers and customers, which means that they collect products at one or more source points and deliver them to the destination points on their way. Of course, the tugger trains need to visit the source points before they visit the destination points [11]. In some cases, they operate on a fixed schedule. The system, therefore, is comparable to a bus system in public transportation [11].

The benefits of using a system of this type include improved efficiency of the overall logistics system and potential substantial savings in shortening the total distance travelled and minimizing the number of vehicles applied, along with remarkable cost advantages related to inventory costs [17]. In this context, the problems of milk-run delivery and distribution of in and out-bound material are usually viewed and formulated as vehicle routing problems (VRP), whose objective is to obtain a minimum-cost route plan to serve a set of customers with known demands, i.e. to assign the items to vehicles that ship them from one depot to another [13, 19]. Consequently, the milk-run problems of component/part/commodity distribution can be classified similarly to the exhaustively studied extensions of the VRP concerning, for instance:

- Capacitated VRP, where the aim is to satisfy the needs of all the customers at different locations by having a given number of vehicles with capacity constraints.
- Consistent VRP, in which the same customers are serviced by the same driver at roughly the same time period over a planning horizon.
- VRP with Time Windows, which is a generalization of the VRP where the service of any customer starts within a given time interval, called a time window [13, 25],
- VRP with Backhauls, also known as the linehaul-backhaul problem, is an extension of the VRP involving both delivery and pick-up points [16],
- VRP with a multi-trip multi-traffic pick-up and delivery problem with time windows and synchronization is a combination of variants of the vehicle routing problem with multiple trips, the vehicle routing problem with time windows, and the vehicle routing problem with pick-up delivery [25].

Besides a huge collection of papers covering different technical problems and addressing issues derived from everyday life practice, there is a large body of methods and problem-solving techniques employed in the course of their modeling and investigation. The modeling frameworks consist of operation research methods (such as linear and nonlinear programming, MLP, computer simulation, and so on) and artificial intelligence methods, such as evolutionary computation (including metaheuristic and stochastic optimization algorithms) [8, 10, 15], and fuzzy-set methods.

The majority of the research in the field of milk-run logistics is devoted to the analysis of the methods of organizing transport processes in ways that minimize the

size of the fleet, the distance travelled (energy consumed), or the space occupied by the infrastructure of a distribution network. The most commonly formulated routing problems are those aimed at maximizing the utilization of fleet capacity, finding the best routing and determining the number of parts to be collected from each supplier on each trip. Other frequently encountered routing problems address the questions of “How to assign certain sequences of stops to certain routes?” and “How to configure tigger trains?” [22]. In practice, many restrictions on facility layout, e.g. one-ways or the radius of the curves/turns, as well as different types of trailer configurations have to be taken into account. Apart from choosing the routes which determine the time schedule, one also has to choose the type and number of parts that must be transported by the different tigger trains to fulfill the orders from various customers. In other words, milk-run scheduling boils down to determining in what time windows parts can be collected from suppliers and delivered to customers along the established routes, so that the cost of transport operations and the size of the inventory in the supply chain are kept at the minimum. In the general case, however, the main point is to simultaneously optimize vehicle routes and dispatch frequency in order to minimize transportation and inventory costs. In that context, the milk-run method seems to be well-suited to solving problems of scheduling and dispatching of inventory in warehouses/supermarkets and production facilities with in-plant transport systems.

Even though the literature addresses a large scope of problems and methods regarding tigger train routing and scheduling, only a limited number of papers are devoted to robust and congestion-free scheduling of a fleet of vehicles subject to in-plant layout constraints. In this respect, the most relevant factors are those which depend on critical, and often unpredictable, traffic congestions which occur when logistics operators allocate too many collecting tasks to the available vehicles, generating unperformed activities due to assumed just-in-time constraints imposed by the time windows of customer services [13, 15]. In focusing on the search for optimal solutions, these studies implicitly assume that there exist admissible solutions, e.g. ones that ensure collision- and/or deadlock-free (congestion-free) flow of concurrent transport processes. In practice, this requires either online updating (revision) of the routing policies used, or prior (offline) planning of congestion-free vehicle routes and schedules. Studies on generating dynamic routing policies are conducted sporadically; even less frequent are investigations of robust routing and scheduling of milk-run traffic, which are, by and large, limited to AGV systems. This is due to the fact that the congestion-avoidance problem, which conditions the existence of admissible solutions, is an NP-hard problem [27] as the necessary and sufficient conditions for deadlock-free execution of concurrent processes are not known. In the absence of these conditions, system analysis (i.e. analysis of the states potentially leading to system deadlocks) has to be carried out which requires the use of laborious and time-consuming computer simulation methods [6, 9].

Consequently, in real-life applications, congestion-avoidance methods (e.g. deadlock prevention) are used, which implement sufficient conditions for the collision-free execution of processes. This means that the time-consuming method of analyzing distribution networks with a view to detecting situations which lead to deadlocks



between concurrent transport flows can be replaced by searching for a synchronization mechanism that would guarantee cyclic execution of these flows.

Methods that are most commonly employed for such purposes include those that use the formalism of max-plus algebra [20, 26] and constraint programming [24]. It should be noted, however, that the possibility of fast implementation of a process-synchronization mechanism (e.g. employed by deadlock-prevention methods) comes at the expense of omitting some of the potentially possible scenarios (e.g. ones that include optimal solutions) for deadlock-free execution of the processes. The shortcomings of the methods providing admissible solutions restrict their implementation in DSS systems, in particular in systems supporting the planning of milk-run traffic flows.

Given this background, our contribution boils down to the assessment of the possibility of using declarative modeling in decision support tools dedicated to prototyping in-plant milk-run traffic systems. This issue, which takes into account the specific character of milk-run systems, has been discussed in our previous work on the leveling of multi-product batch production flows [5] and a declarative modeling framework for routing and scheduling of Unmanned Aerial Vehicles [2].

### 3 Modeling

The example of a multi-item batch flow production system presented in this section illustrates typical cases of problems of analysis and synthesis of milk-run in-plant distribution networks. In one of the cases (an analysis problem), two variants of traffic flow organization are analyzed, in which two tugger trains moving along different routes serve disjoint sets of workstations. In the first variant, all workstations are serviced in the same cycle, i.e. the delivery time windows and the pick-up time windows are spaced at the same intervals in the production cycle (with a period  $T$ ). In the second variant, a change in the length of the cycle of one of the tugger trains (still moving along the same route as before the change) leads to train deadlocks and collisions within period  $T$ . This observation spurred further experiments focused on the problems of synthesis and analysis of milk-run systems, in which different groups of workstations with different delivery/pick-up time window densities are serviced by different tugger trains. The goal of the synthesis problem was to find new, collision-free routes for the available tugger trains. The analysis problem was formulated to obtain solutions which would allow to modify tugger train velocities (without changing their routes), so that the individual tugger train trips could be executed over the entire period  $T$  in time slots of quasi-same periods. In other words, the goal was to seek solutions which returned cyclic schedules consisting of quasi-cyclic partial schedules.

### 3.1 Motivating Example

Consider the milk-run system in which two production flows of products  $J_i$  (job  $i$ ) are executed (batch size  $B_i = 1$ , see Fig. 1a). The system consists of a warehouse  $M_9$ , a supermarket  $M_8$ , and seven workstations  $M_1$ – $M_7$ . The technological route for product  $J_1$  is marked in violet (route:  $M_1, M_2, M_5, M_7$ ) and the route for product  $J_2$  is marked in magenta (route:  $M_3, M_4, M_6$ ). The so-called complex operations  $O_{i,j}$  (i.e. processes that are made up of elementary operations executed by the individual workstations of a production cell) have the following times:

- $t_{1,1} = 250$  u.t. (units of time),  $t_{1,2} = 550$  u.t.,  $t_{1,3} = 600$  u.t. and  $t_{1,4} = 420$  u.t., for job  $J_1$ ;
- $t_{2,1} = 300$  u.t.,  $t_{2,2} = 600$  u.t.,  $t_{2,3} = 430$  u.t., for job  $J_2$ ;

which determine the value of production takt time  $TP_1 = TP_2 = 600$  u.t. governed by bottleneck resources  $M_5, M_6$ .

A Gantt chart of production flow in the investigated system is shown in Fig. 2. The time windows for the delivery/pick-up of components to workstations  $M_1$ – $M_7$ , determined by the production schedule from Fig. 1b are as follows:

- $PR_1 = (TD_{1,1} = [30, 210], TD_{1,2} = [280, 460], TC_{1,3} = [830, 1010],$   
 $TC_{1,4} = [2030, 2210])$
- $PR_2 = (TD_{2,1} = [470, 650], TD_{2,2} = [770, 950], TD_{2,3} = [1980, 2160]).$

For example time window  $TD_{1,2} = [280, 460]$  is the period of time in which the components should be delivered to workstation  $M_2$  in order to execute operation  $O_{1,2}$ . It is easy to note that in both cases, the production takt times  $TP_1 = TP_2 = 600$  u.t. are same and are determined by timely (occurring within the given time windows) delivery/pick-up of intermediate components/finished products to/from the given docking stations .

In other words, the production flow schedule (shown in Fig. 1b) determines the schedule of visits to the individual tugger train docking stations (sequences  $PR_1, PR_2$ ).

A fleet composed of two tugger trains  $TT_1, TT_2$  (marked in orange and green, respectively, see Fig. 2) is used to service seven workstations  $M_1$ – $M_7$ , the supermarket ( $M_8$ ) and the warehouse ( $M_9$ ), while providing dedicated material pick-up/delivery operations. Tugger train  $TT_1$ , which follows the transportation route designated by docking stations  $M_1$  –  $M_5$ , delivers intermediate components from the supermarket ( $M_8$ ) to the relevant workstations.

In turn, tugger train  $TT_2$ , which follows the transportation route designated by docking stations  $M_6, M_7$ , picks-up finished products from the relevant set of workstations to deliver them to the warehouse ( $M_9$ ). All requests must be met within the given time windows  $PR_1, PR_2$ . In other words, the delivery operations must be completed within a period of 180 u.t. before job operation start time, and the pick-up operations should be started within the period of 180 u.t. after the job operation completion time.

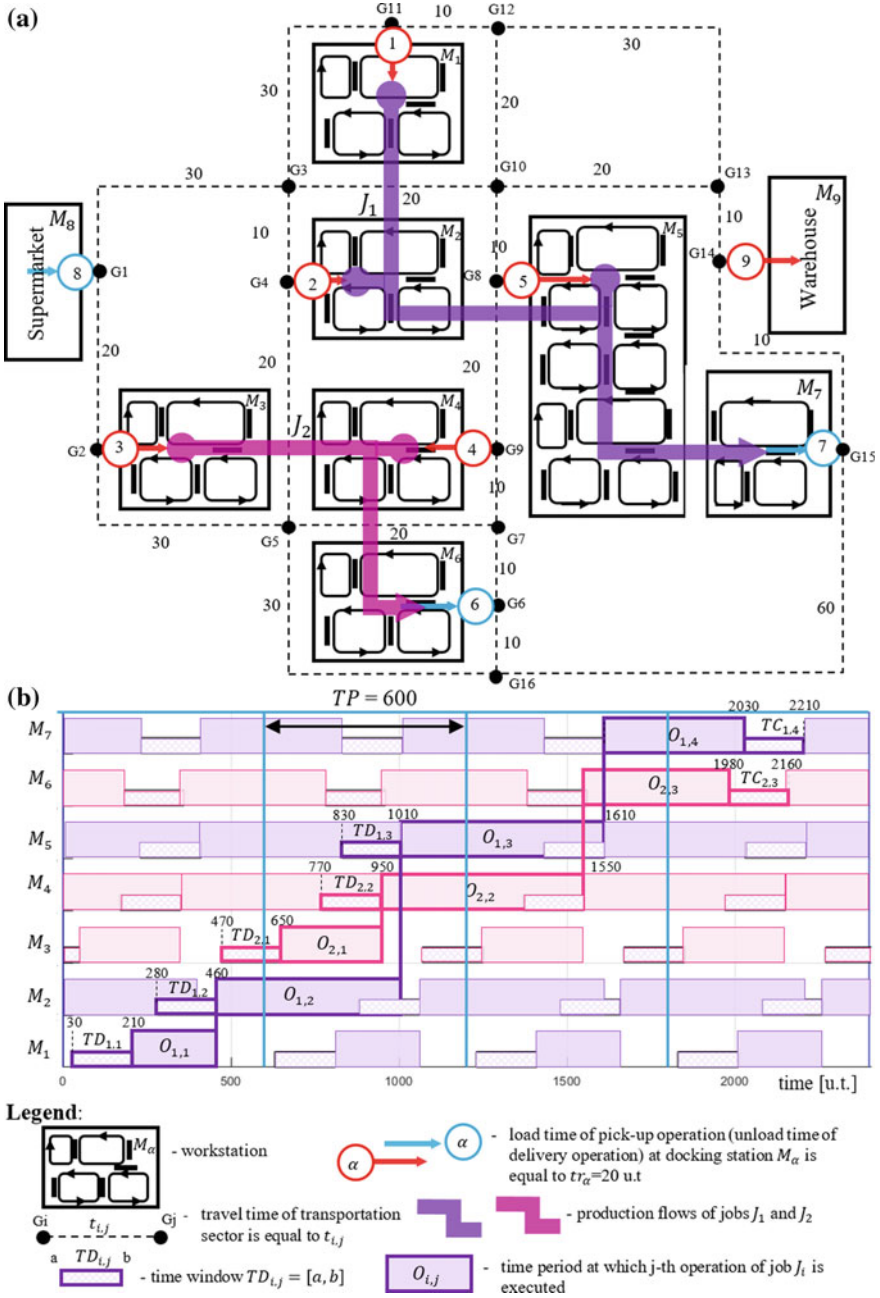
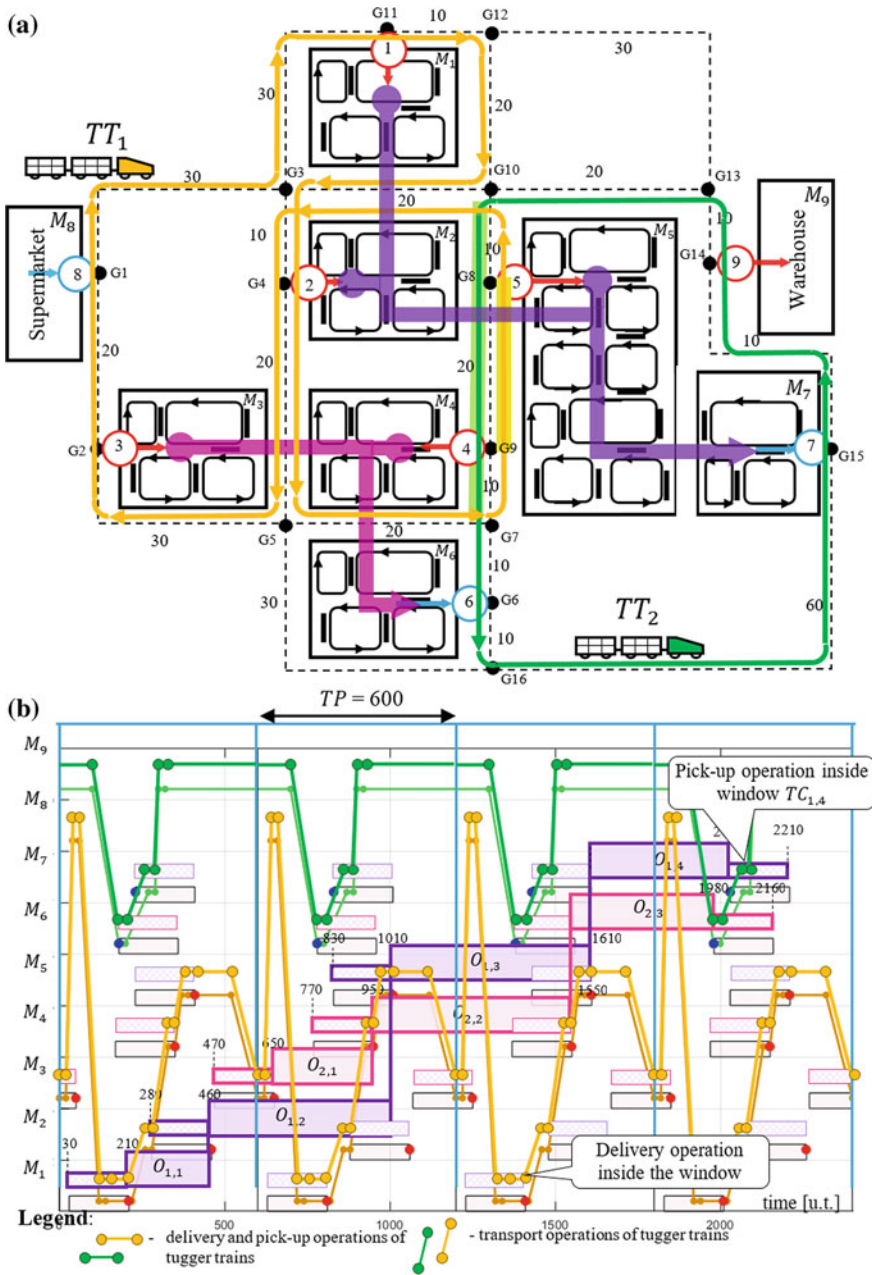


Fig. 1 Milk-run system under consideration (a) and a Gantt chart of production flow (b)



**Fig. 2** Tigger trains (a) which guarantee a production flows schedule with takt time  $TP = 600$  u.t. shown in the Gantt chart (b)

Consequently, the delivery and pick-up time windows (following from production flows) can be thought of as interacting, though independently flowing streams of deliveries servicing both part supply and product pick-up. Assuming that:

- tugger train  $TT_1$  is used to transport components from the supermarket to docking stations  $M_1$ – $M_5$  (in cycles with a period further denoted by  $\text{TS}$ ),
- tugger train  $TT_2$  is used to transport products to the warehouse from docking stations  $M_6, M_7$  (in cycles with a period further denoted by  $\text{TS}$ ),
- the travel times along transportation sections, which are the same for each tugger train, are known in advance (see Fig. 1a),
- the times of loading and unloading operations are the same for each docking station (see Fig. 1a),
- at a given moment, a docking station (transport section) can only be occupied by one tugger train,
- a resource (e.g. a warehouse, a supermarket, a workstation) is serviced by a single docking station  $M_\alpha$ ,
- each request should be granted within the given delivery/pick-up time windows  $PR_1, PR_2$  (see Fig. 1b),

the following question can be considered: *Do there exist, for the given fleet of tugger trains, routes that allow to deliver/pick up items to and from the given docking stations in time windows  $PR_1, PR_2$ ?*

Examples of answers to this question are provided by the solutions shown in Fig. 2. These solutions have been obtained assuming that transport between workstations, e.g.  $M_1$ – $M_2$ ;  $M_2$ – $M_5$ ;  $M_5$ – $M_7$ ;  $M_3$ – $M_4$  and  $M_4$ – $M_6$  is supported by an overhead transport system. The tugger train routes ( $\pi_1$  and  $\pi_2$ ) obtained are (Fig. 2a):

- $\pi_1 = (M_8, M_1, M_2, M_4, M_5, M_3)$ —orange line in Fig. 2a.
- $\pi_2 = (M_9, M_6, M_7)$ —green line in Fig. 2a.

A Gantt chart showing how delivery and pick-up operations are executed in a system implementing this type of solution is presented in Fig. 2b. It is easy to see that all transport operations are executed within appropriate time windows  $PR_1$  and  $PR_2$ . In other words, the obtained tugger train schedule (determined by routes  $\pi_1$  and  $\pi_2$ ) allow to maintain the required production takt time  $TP = \text{TP} = \text{TS} = 600$  u.t.

The example refers to a situation in which all workstations operate at the same takt time (600 u.t.). In practice, an assumption like this is hard to fulfill. In special cases, each workstation may have its own operation takt time.

Let us consider a situation in which some workstations ( $M_1$ – $M_5, M_8$ ) work at takt time  $\text{TP} = 600$  u.t. and the rest ( $M_6, M_7, M_9$ ) operate at takt time  $\text{TS} = 800$  u.t. Let us also assume that the routes of tugger trains  $TT_1, TT_2$  are the same as previously:  $\pi_1 = (M_8, M_1, M_2, M_4, M_5, M_3)$  and  $\pi_2 = (M_9, M_6, M_7)$ , respectively, (see Fig. 2a). In this context, the delivery requests following sequence  $\pi_1$  are serviced at takt time  $\text{TP} = 600$  u.t. by tugger train  $TT_1$ , while pick-up requests following sequence  $\pi_2$  are serviced at takt time  $\text{TS} = 800$  u.t. by tugger train  $TT_2$ . For this kind of assumptions, the following question can be considered:

Does there exist a tugger train schedule (for routes  $\pi_1$  and  $\pi_2$ ) which guarantees timely (i.e. performed within the given time windows) delivery/pick-up of intermediate components/finished products to/from the given docking stations?

Examples of solutions are shown in Fig. 3. It is easy to note that the assumed takt times determine the period of the cyclic milk-run system schedule, which is equal to the least common multiple  $T = LCM(TP_1, TP_2) = LCM(600, 800) = 2400$  u.t. This period covers four time sub-windows of 600 u.t. and three time sub-windows of 800 u.t. (see Fig. 3). It should be noted that in some time windows (the second time sub-window in the grey-shaded interval) two tugger trains travelling through a shared critical region composed of a transport section ( $G_9-G_8$ ) are in a deadlock.

This observation is in line with the assumption that a critical region consists of individual carriageway sections which must not be utilized by more than one tugger train at a time. Note that, in the considered case, tugger train  $TT_1$  moves from  $M_4$  to  $M_5$  along ( $G_9-G_8$ ) (highlighted by a bold green line) while tugger  $TT_2$  moves from  $M_9$  to  $M_6$  along a sequence of sections: ( $G_{14}-G_{13}$ ), ( $G_{13}-G_{10}$ ), ( $G_{10}-G_8$ ), ( $G_8-G_9$ ), ( $G_9-G_7$ ), ( $G_7-G_6$ ) (highlighted by a yellow line).

Occurrence of a deadlock in sector ( $G_9-G_8$ ) shown in Fig. 3 eliminates this solution from further consideration. Among the remaining routing variants, there are no solutions that would enable deadlock-free movement of the considered tugger

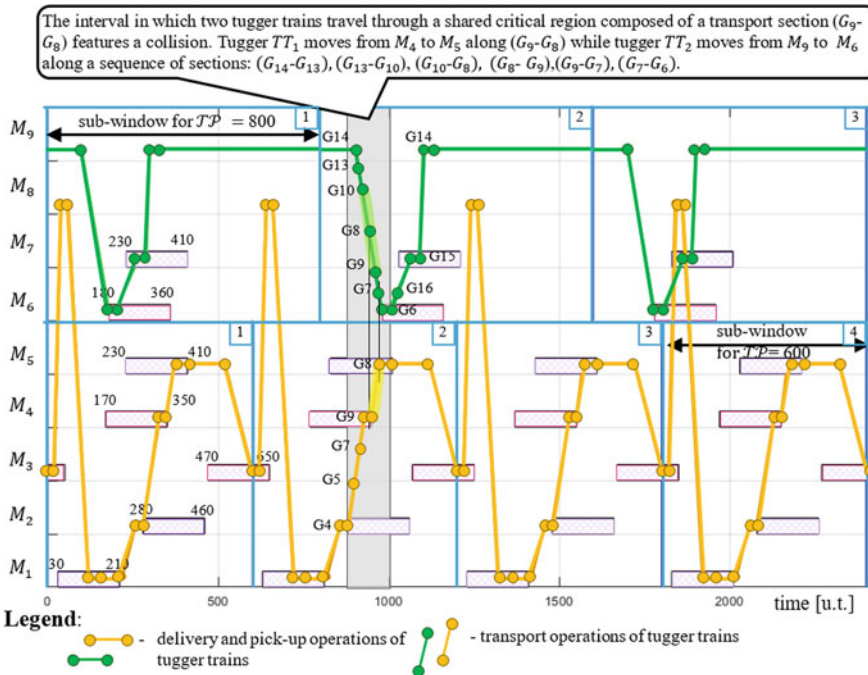


Fig. 3 Gantt chart of production schedule with distinguished critical region

trains in the system from Fig. 2 along routes  $\pi_1 = (M_8, M_1, M_2, M_4, M_5, M_3)$  and  $\pi_2 = (M_9, M_6, M_7)$ .

What remains, then, is to look for other routes  $\pi_1$  and  $\pi_2$  (or possibly another fleet of trains) which would guarantee timely delivery/pick-up of items to/from workstations without leading to deadlocks of trains servicing production flows simultaneously, but at different takt times.

One of the possible solutions to the problem of synthesis formulated in this way is a set of routes  $\Pi = \{\pi_1, \pi_2\}$ :

- $\pi_1 = (M_8, M_1, M_5, M_4, M_2, M_3)$ —orange line in Fig. 4a,
- $\pi_2 = (M_9, M_6, M_7)$ —green line in Fig. 4a,

which guarantees deadlock-free movement of trains within the entire interval with period  $T = 2400$  u.t.; see the tugger train schedule from Fig. 4b. In this solution, there is no deadlock, however, there is a danger of collision.

A situation of this type is shown in the grey-shaded time window in Fig. 4b, in which train  $TT_1$  unloaded at stop  $M_4$  is overtaken at node  $G_9$  by train  $TT_2$  traveling between stops  $M_9$ – $M_6$ . When the stops are not located in specially designed loading/unloading bays, collisions may occur.

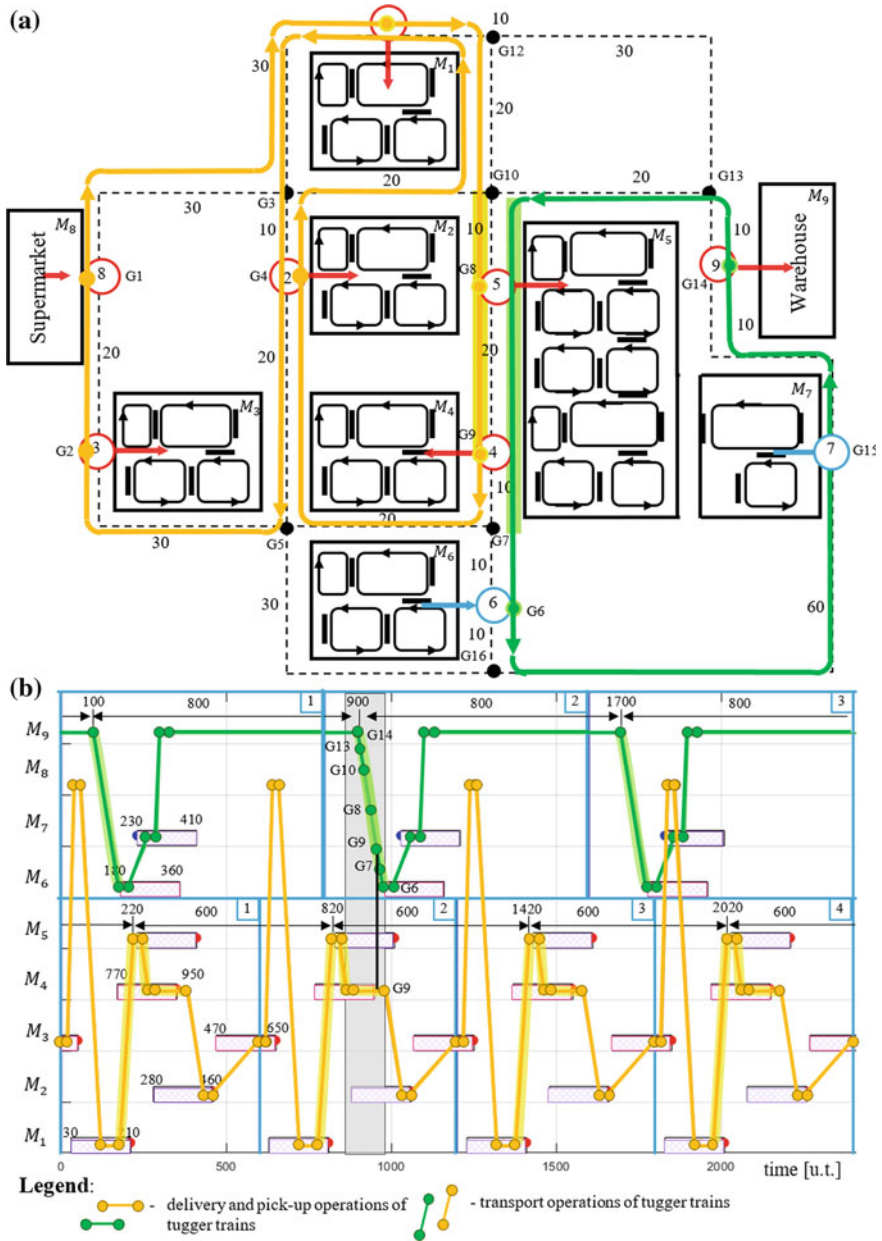
Figure 5 shows a solution in which there are no collisions or deadlocks. This solution features a set of routes along which tugger trains travel periodically at period  $T = 2400$  u.t. In contrast to the previous solutions, in this routing variant train trips are quasi-periodic. In other words, train travel cycles do not coincide with the time sub-windows determined by takt times  $\mathbb{T}P = 600$  u.t. and  $\mathbb{T}S = 800$  u.t.).

Tugger train schedules for these windows are shifted relative to each other by 10 u.t. and 20 u.t., respectively — see Fig. 5b. The possibility that this solution provides of having different tugger train schedules for the successive time sub-windows allows one to choose such delivery/pick-up operation start/end times which enable simultaneous movement of trains without deadlocks and collisions.

The procedure described above, in which analysis of system behavior (evaluation of collision-free behavior) and synthesis of its parameters (synthesis of routes, fleet size, etc.) are performed alternately corresponds well with the proposed decision support methodology described in more detail in Sect. 4.

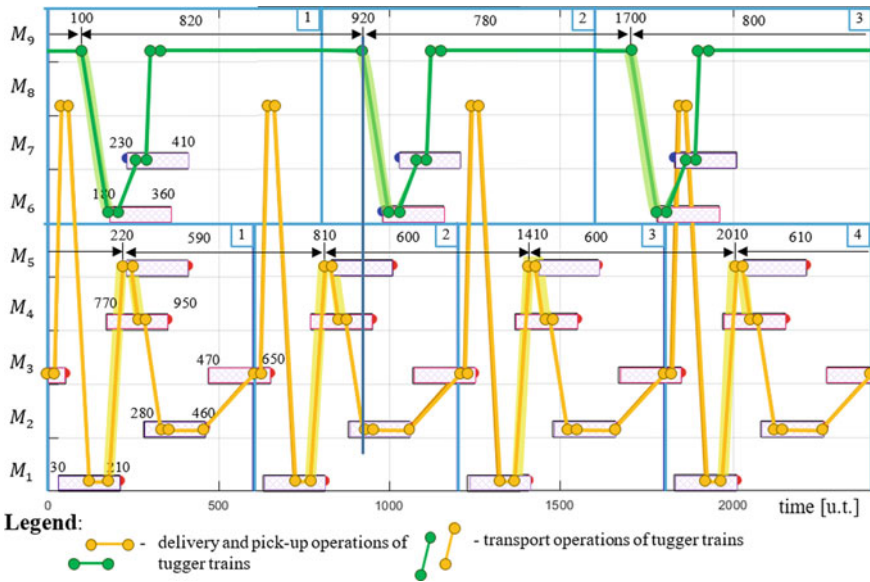
### 3.2 Problem Formulation

The questions considered in the previous section are specific cases of the problems of synthesis and analysis of a multi-item flow production system. These problems involve either evaluation (analysis) of the attainability of an expected behavior, e.g. the assessment whether, for a given layout (routes), there exists a schedule that can guarantee collision-free movement of trains, or synthesis of a layout that will guarantee the expected behavior, e.g. the assessment whether there exist train routes along which trains can travel in a collision-free manner?



**Fig. 4** Routes (a) and Gantt chart of a routing/schedule (b) guaranteeing deadlock-free movement of tugger trains





**Fig. 5** Gantt chart of a schedule guaranteeing deadlock and collision-free movement of tigger trains

In the case of the class of milk-run systems under consideration, these problems can be formulated as constraint satisfaction problems. To this end, the following model with two different production takt times ( $TP$  for pick-up operations and  $TS$  delivery operations) was adopted:

**Symbols:**

- $J_i$  job  $i$  (production process);
- $O_{i,q}$  operation  $q$  of  $J_i$ ;
- $TT_v$   $v$ th transport process (performed by the  $v$ th tigger train);
- $T$  the period of a production cycle;
- $M_\alpha$   $\alpha$ th docking station (associated with the warehouse, the supermarket, the workstation);
- $o_\alpha$   $\alpha$ th transport operation (operation of delivery/pick-up of a part/product to/from a docking station);
- $b_\alpha$  index of transport operation which precedes  $o_\alpha$ ;
- $f_\alpha$  index of transport operation which follows  $o_\alpha$ .

**Parameters:**

- $l$  number of transport means;
- $\omega$  number of docking stations;
- $tr_\alpha$  operation time of  $o_\alpha$ ;
- $d_{\alpha,\beta}$  travel time between docking station  $M_\alpha$  and docking station  $M_\beta$ ;

TP	assumed value of production takt time for pick-up requests;
TS	assumed value of production takt time for delivery requests;
$\mathbb{K}_{\varepsilon\beta-\lambda\gamma}$	intersection indicator. $\mathbb{K}_{\varepsilon\beta-\lambda\gamma} = 1$ if $(o_\varepsilon, o_\beta)$ and $(o_\lambda, o_\gamma)$ are pairs of consecutive operations, and the path connecting docking stations $M_\varepsilon, M_\beta$ , are executed crosses the path that connects the docking stations $M_\lambda, M_\gamma$ ; $\mathbb{K}_{\varepsilon\beta-\lambda\gamma} = 0$ in the remaining cases.

### Sets and sequences:

$TT$	set of transport means $TT_v$ (tugger trains);
$M$	set of docking stations $M_\alpha$ ;
$J$	set of jobs $J_i$ , (production processes);
$\mathbb{O}$	set of transport operations $o_\alpha$ ;
$\mathbb{P}$	set of pick-up operations, $\mathbb{P} \subseteq \mathbb{O}$ ;
$\mathbb{S}$	set of delivery operations, $\mathbb{S} \subseteq \mathbb{O}$ ;
$PR_i$	sequence of delivery/pick-up time windows $TD_{i,q}, TC_{i,q}$ of job $J_i$ ;
$PR$	set of sequences $PR_i$ ;
$B$	sequence of predecessor indices of delivery operations, $B = (b_1, \dots, b_\alpha, \dots, b_\omega), b_\alpha \in \{0, \dots, \omega\}$ ;
$F$	sequence of successor indices of delivery operations, $F = (f_1, \dots, f_\alpha, \dots, f_\omega), f_\alpha \in \{1, \dots, \omega\}$ ;
$\pi_v$	$\pi_v = (M_{v_1}, \dots, M_{v_i}, M_{v_{i+1}}, \dots, M_{v_\mu})$ where: $v_{i+1} = f_{v_i}$ for $i = 1, \dots, \mu - 1$ and $v_1 = f_{v_\mu}$ —route of $v$ th transport process following sequence (determined by $F$ ) of docking stations serviced by tugger train $TT_v$ ;
$\Pi$	set of routes $\pi_v$ .

### Variables:

$xt_\alpha$	start time of operation $o_\alpha$ at the $\alpha$ th docking station
$yt_\alpha$	end time of operation $o_\alpha$ ;
$xs_\alpha$	moment the resource occupied by a tugger train is released after completion of operation $o_\alpha$ ;
$b_\alpha$	index of the transport operation preceding operation $o_\alpha$ (operations $o_{b_\alpha}$ and $o_\alpha$ are executed by the same tugger train); $b_\alpha = 0$ means that $o_\alpha$ is the first operation in the system cycle;
$f_\alpha$	index of the transport operation following $o_\alpha$ , (operations $o_\alpha$ and $o_{f_\alpha}$ are executed by the same tugger train).

### Constraints:

1. transport process operations:

$$yt_\alpha = xt_\alpha + tr_\alpha, \alpha = 1, 2, \dots, \omega, \quad (1)$$

$$b_\alpha = 0, \forall \alpha \in BS, BS \subseteq BI = \{1, 2, \dots, \omega\}, |BS| = l \quad (2)$$

$$b_\alpha \neq b_\beta \forall \alpha, \beta \in BI \setminus BS, \alpha \neq \beta, \quad (3)$$

$$f_\alpha \neq f_\beta \forall \alpha, \beta \in BI, \alpha \neq \beta, \quad (4)$$

$$(b_\alpha = \beta) \Rightarrow (f_\beta = \alpha), \forall b_\alpha \neq 0, \quad (5)$$

$$[(b_\alpha = \beta) \wedge (b_\beta \neq 0)] \Rightarrow (yt_\beta + d_{\beta, \alpha} \leq xt_\alpha), \alpha, \beta = 1, 2, \dots, \omega, \quad (6)$$

$$xs_\alpha \geq yt_\alpha, \alpha = 1, 2, \dots, \omega, \quad (7)$$

$$[(f_\alpha = \beta) \wedge (b_\beta \neq 0)] \Rightarrow (xs_\alpha = xt_\beta - d_{\alpha, \beta}), \alpha, \beta = 1, 2, \dots, \omega, \quad (8)$$

$$[(f_\alpha = \beta) \wedge (b_\beta = 0)] \Rightarrow (yt_\alpha + d_{\alpha, \beta} \leq xt_\beta + \mathbb{TP}), \forall o_\alpha, o_\beta \in \mathbb{P}, \quad (9)$$

$$[(f_\alpha = \beta) \wedge (b_\beta = 0)] \Rightarrow (yt_\alpha + d_{\alpha, \beta} \leq xt_\beta + \mathbb{TS}), \forall o_\alpha, o_\beta \in \mathbb{S}, \quad (10)$$

$$[(f_\alpha = \beta) \wedge (b_\beta = 0)] \Rightarrow (xs_\alpha = xt_\beta - d_{\alpha, \beta} + \mathbb{TP}), \forall o_\alpha, o_\beta \in \mathbb{P}, \quad (11)$$

$$[(f_\alpha = \beta) \wedge (b_\beta = 0)] \Rightarrow (xs_\alpha = xt_\beta - d_{\alpha, \beta} + \mathbb{TS}), \forall o_\alpha, o_\beta \in \mathbb{S}, \quad (12)$$

## 2. transport operations and production requests

$$[TC_{i,q}] \leq xt_\alpha \leq [TC_{i,q}], \forall o_\alpha \in \mathbb{P}, o_\alpha \text{ is associated with } O_{i,q} \quad (13)$$

$$[TD_{i,q}] \leq yt_\alpha \leq [TD_{i,q}], \forall o_\alpha \in \mathbb{S}, o_\alpha \text{ is associated with } O_{i,q} \quad (14)$$

## 3. collision- and deadlock-free traffic

$$\left( \mathbb{K}_{\varepsilon\beta-\lambda\gamma} = 1 \right) \Rightarrow \left( (xt_\beta + k \cdot TP(\beta) \leq \text{mod} \{xs_\lambda, TP(\lambda)\} + l \cdot TP(\lambda)) \vee \right. \\ \left. \vee (xt_\gamma + l \cdot TP(\gamma) \leq \text{mod} \{xs_\varepsilon, TP(\varepsilon)\} + k \cdot TP(\varepsilon)) \right) \quad (15)$$

$$\varepsilon, \lambda, \beta, \gamma = 1, 2, \dots, \omega,$$

$$k, l = 1, 2, \dots, q,$$

$$q = \max \left\{ \frac{LCM(\mathbb{TS}, \mathbb{TP})}{\mathbb{TP}}, \frac{LCM(\mathbb{TS}, \mathbb{TP})}{\mathbb{TS}} \right\},$$

where:

$$TP(\alpha) = \begin{cases} \mathbb{TP} & \text{if } o_\alpha \in \mathbb{P}, \\ \mathbb{TS} & \text{if } o_\alpha \in \mathbb{S}. \end{cases}$$

## Analysis problem

The problem of analysis based on the proposed model can be formulated as (16)

$$CS_A = ((X, \mathbb{D}), \mathbb{C}_A), \quad (16)$$

where:

- $X$  decision variables, a cyclic schedule of process operations executed in milk-run cycles:  $X = (X', Xs')$ , where:  $X' = (xt_\alpha | \alpha = 1 \dots \omega)$ ,  $Xs' = (xs_\alpha | \alpha = 1 \dots \omega)$ ,
- $\mathbb{D}$  a finite set of decision variable domains  $X$ :

$$x_\alpha \in \{0, \dots, T\}, \quad xs_\alpha \in \{0, \dots, 2T\},$$

- $\mathbb{C}$  a set of constraints specifying the relationships between the operations of the processes implemented in milk-run cycles and the processes executed along the technological routes of individual products:
- constraints on the order (1)–(8) and cyclic execution (9)–(12) of transport operations;
  - constraints which guarantee timely completion of requests (13)–(14);
  - constraints which ensure collision- and deadlock-free execution of milk-run flows (15).

Schedule  $X$  of problem (16) determines a tugger train timetable which guarantees timely delivery (in conformity with the starting times of operations of processes executed along the technological routes of individual products) of materials to workstations. This means that, assuming that the set of tugger trains  $TT$  travelling in milk-run cycles, their routes  $\mathcal{D}$ , process execution times  $tr_\alpha$  and process flow times  $d_{\alpha,\beta}$  are known, and also known are delivery/pick-up time windows  $PR$ , to solve problem  $CS_A$  (16) one only needs to determine such values of decision variables  $X$ , for which all constraints given in set  $\mathbb{C}_A$  (1)–(15) are satisfied. Consequently, any admissible solution being a cyclic schedule provides a tugger trains fleet timetable, encompassing blockage-free execution of milk-run material delivery processes.

### Synthesis Problem

The synthesis problem is given by (17):

$$CS_S = ((\{X, B, F\}, \mathbb{D}), \mathbb{C}_S), \quad (17)$$

where:

- $\{X, B, F\}$  a set of decision variables including:

- $X$  decision variables, a cyclic schedule of process operations executed in milk-run cycles:  $X = (X', Xs')$ , where:  $X' = (xt_\alpha | \alpha = 1 \dots \omega)$ ,  $Xs' = (xs_\alpha | \alpha = 1 \dots \omega)$ .
- $B, F$  sequences specifying the order in which operations  $o_\alpha$  are executed by the successive processes run in milk-run loops (i.e. processes of set  $P$ ). Each pair  $(B, F)$  is matched with exactly one pair from the set of routes  $\Pi$ .

$\mathbb{D}$  a finite set of decision variable domains  $\{X, B, F\}$ :

$$x_\alpha \in \{0, \dots, T\}, \quad x_{s_\alpha} \in \{0, \dots, 2T\}, \quad b_\alpha \in \{0, \dots, \omega\}, \quad f_\alpha \in \{1, \dots, \omega\},$$

$\mathbb{C}_S$  a set of constraints specifying the relationships between the operations of the processes implemented in milk-run cycles and the processes executed along the technological routes of individual products:

- constraints on the order (1)–(8) and cyclic execution (9)–(12) of transport operations;
- constraints which guarantee timely completion of requests (13)–(14);
- constraints which ensure collision- and deadlock-free execution of milk-run flows (15).

Decision variables  $B, F$  correspond to the structure parameters (routes of tugger trains  $\mathcal{D}$ ) which are to guarantee the existence of schedule  $X$  that will enable timely delivery of materials to workstations (in accordance with the delivery/pick-up time windows  $PR$ ). In other words, assuming that the set of tugger trains  $TT$  travelling in milk-run cycles, execution times  $t_\lambda$ , process flow times  $d_{\lambda,\beta}$ , and also time windows  $PR$  are all known, to solve problem  $CS_S$  (17), it is enough to find such values (determined by domains  $\mathbb{D}$ ) of decision variables  $B, F$  (routes of local processes  $\mathcal{D}$ ) and  $X$ , for which all constraints given in set  $\mathbb{C}_S$  (1)–(15) are satisfied.

### 3.3 Recursive Constraint Satisfaction Problem

Computer implementation [2] of the problems of synthesis (16) and analysis (17) of milk-run systems in constraint programming environments such as Oz Mozart, ILOG, and ECL<sup>i</sup>PS<sup>E</sup>, etc [3, 24] allows to determine routes  $\Pi$  and delivery schedules  $X$ , on the condition that train trip cycles coincide with the periods of the time sub-window, i.e.,  $TP_1 = 600$  u.t. and  $TP_2 = 800$  u.t.—see Figs. 3b and 4b.

However, as the example in Fig. 5 clearly shows, it is also possible to search for deadlock-free and collision-free solutions that do not meet this condition. Such solutions yield quasi-periodic train trip cycles which do not overlap with the time sub-window periods. The observed quasi-cyclicity of train schedules, i.e. cyclically repeating schedules with period  $T$ , entails the need to decompose the above-mentioned CSPs into time sub-windows and to determine the decision variables sought recursively for the successive CSPs defined by the time sub-windows occurring in period  $T$ .

Figure 7 shows a schema of interactive planning of a milk-run delivery system, the first two stages of which, corresponding, respectively, to an analysis and a synthesis problem, are illustrated by the examples discussed in point Sect. 3.1 (see Figs. 4 and 5). The third stage refers to another analysis problem in which trains  $TT_1$  and  $TT_2$ , moving along previously designated routes  $\pi_1 = (M_8, M_1, M_5, M_4, M_2, M_3)$  and

$\pi_2 = (M_9, M_6, M_7)$ , respectively, travel at a double speed. The solution sought at this stage should guarantee cyclic movement of tugger trains (overlapping with time sub-window periods) in the considered period  $T$ .

Let us focus on the synthesis problem, first. Taking into account the above mentioned assumptions, an appropriate Constraint Satisfaction Problem (CSP) can be formulated as the following recursive scheme:

$$CS_S(l) = (CS'_S(l) \cup X(l), \mathbb{D}(l), \mathbb{C}_S(l)), \quad (18)$$

where:

$l$  is number of assumed time sub-windows;  
 $CS_S(1)$  is a CSP formulated for the first time sub-window,

$$CS_S(1) = (\{B, F, X(1)\}, \mathbb{D}(1), \mathbb{C}_S(1));$$

$\{B, F, X(1)\}$  is the solution to this problem while following decision variables satisfy the constraints  $\mathcal{C}_S(1)$ ;

$X(l)$   $X(l) = (X'(l), Xs'(l))$ , where:  $X'(l) = (xt_\alpha(l)|\alpha = 1 \dots \omega)$  and  $Xs'(l) = (xs_\alpha(l)|\alpha = 1 \dots \omega)$  are the schedules with moments  $xt_\alpha$  and  $xs_\alpha$  for the  $l$ th time sub-window;

$\mathbb{D}_l$  is a discrete finite set of domains of variables  $B, F, X(l)$ ;

$\mathbb{C}_S(l)$  is a set of constraints describing constraints  $\mathcal{C}_S(1)$ –(15) in the  $l$ th time sub-window (in particular, constraints describing different windows  $PR(l)$  in different time sub-windows, see Fig. 6).

To solve the  $CS_S(l)$  problem (18), the sets of values of decision variables  $\mathbb{V}(l)$  determining tugger train routes  $B, F$  and schedules  $X(l)$  from the  $l$ th time sub-window (i.e. delivery operation schedules), for which all the constraints  $\mathbb{C}_S(l)$  are satisfied, have to be deduced from the preceding problem  $CS_S(l - 1)$  while taking into account the different delivery/pick-up time windows  $PR(l)$  occurring in the  $l$ th time sub-window.

In turn, to solve the  $CS_S(l - 1)$  problem, set  $\mathbb{V}(l - 1)$  from the  $l - 1$ th time sub-window, for which all the constraints  $CS_S(l - 1)$  are satisfied, have to be deduced from the preceding problem  $CS_S(l - 2)$ , while taking into account the different time windows  $PR(l - 2)$  occurring in the  $l - 2$ th time sub-window. And so on, up to

$$CS_S(2) = (CS'_S(2) \cup X(2), \mathbb{D}(2), \mathbb{C}_S(2)),$$

where

$$CS_S(1) = (\{B, F, X(1)\}, \mathbb{D}(1), \mathbb{C}_S(1)).$$

In this context, the  $CS_S(l)$  problem integrates the issues of tugger train routing ( $B, F$ ) and tugger train schedules  $X(1), \dots, X(l)$ . This strategy is illustrated in Fig. 6 using the example of the schedule from Fig. 5.

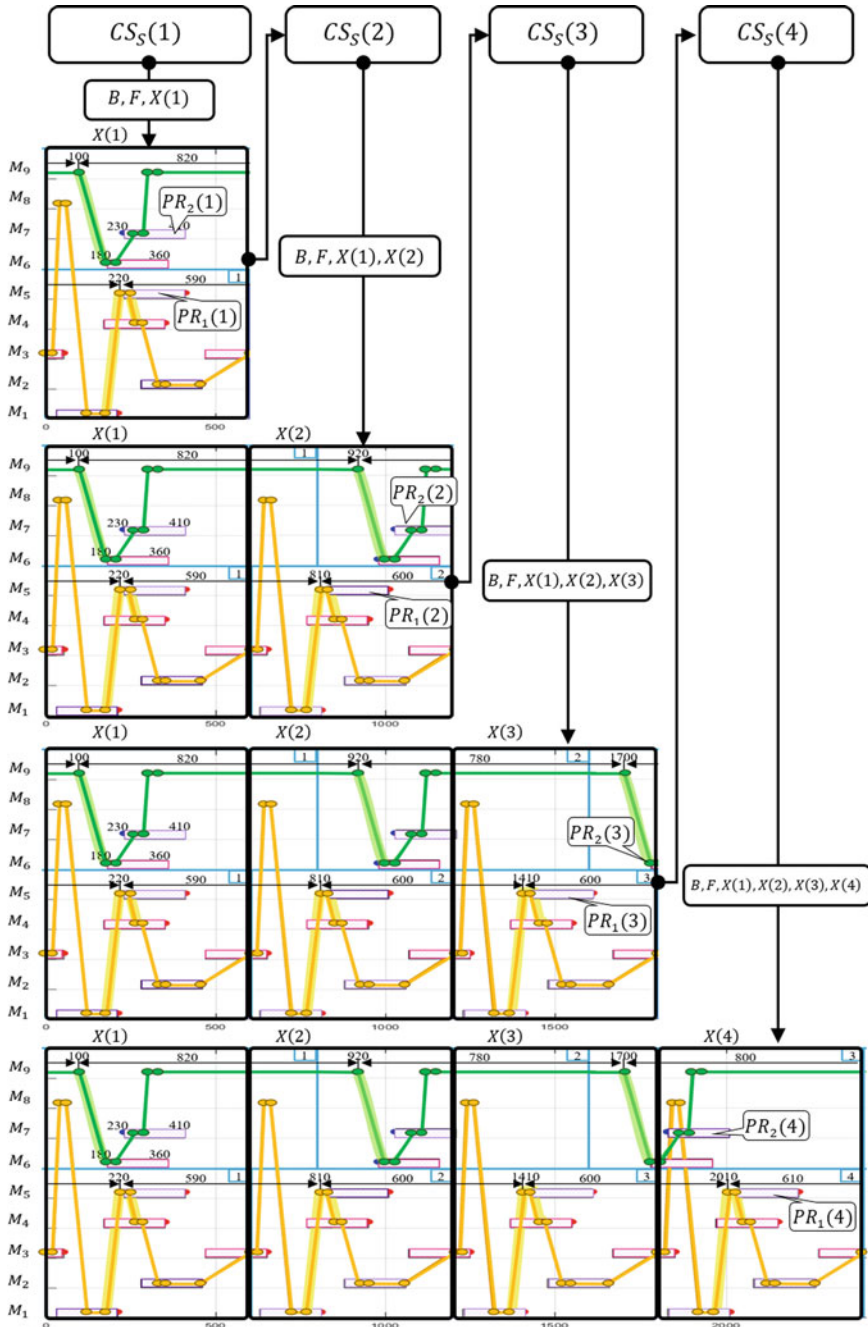


Fig. 6 A recursive schema used for calculation of the schedule from Fig. 5

The recursive formulation of problem (18) is based on the example of a synthesis problem  $CS_S$  (16); an analogous recursive model can also be built for an analysis problem  $CS_A$  (17).

## 4 Method

A computer implementation of the recursive CSP model (18) can be used for the prototyping of delivery/pick-up cycles in a milk-run system. Routes  $\Pi$  and schedules  $X(1), \dots, X(l)$ , are planned, each time, by solving a dedicated synthesis or analysis problem, as specified by access to input data characterizing the structural and functional parameters of the production system being modeled (see Fig. 7).

The strategy of searching for a solution resembles a game situation. The goal of this game is to look for such elements of the system's structure, e.g. routes, resource limits, costs, etc., which either boost the given set of criteria, or suffice to achieve the given values of the set of criteria being considered, e.g. timeliness, deadlock- and collision-free traffic, resource utilization rate, etc. This means that, in the former case, one seeks for such an organization of the structure of the milk-run system, which will boost the given system-performance criteria; in the latter case, one looks for such a structure (e.g. an admissible structure) which allows to achieve the expected values of the parameters characterizing the behavior of the system. Under the action scenarios (variants) for the first stage, one arbitrarily determines the values of selected parameters of the system's structure and evaluates the effect of the changes introduced on the values of selected system performance evaluation criteria. Under scenarios for the second stage, one determines the values of the selected evaluation criteria and checks whether, in the given ranges of variability of the parameters describing the structure of the distribution system, there exist values of these parameters which guarantee the fulfilment of the adopted system performance criteria.

The purpose of the decision support process proposed above is to look for answers to one of the questions presented in Sect. 3.1. This search boils down, each time, to solving a suitable constraint satisfaction problem. The problems selected in this way are solved in the Oz Mozart, ILOG, and ECL<sup>i</sup>PS<sup>E</sup> environments.

If for a given instance of a problem the resulting set of admissible solutions is non-empty, then each solution that belongs to this set includes:

- period  $T$ , a sequence of pick-up/delivery deadlines determined by delivery/pick-up time windows  $PR$ ,
- sequences describing the order of delivery operations  $B, F$ , determining routes  $\Pi$
- sequence of starting times of delivery operations  $X$ ,
- sequence of release times of resources occupied by delivery operations  $Xs$ .

An example illustrating the application of the presented approach to the cases from Figs. 4 and 5 is described in Fig. 7.



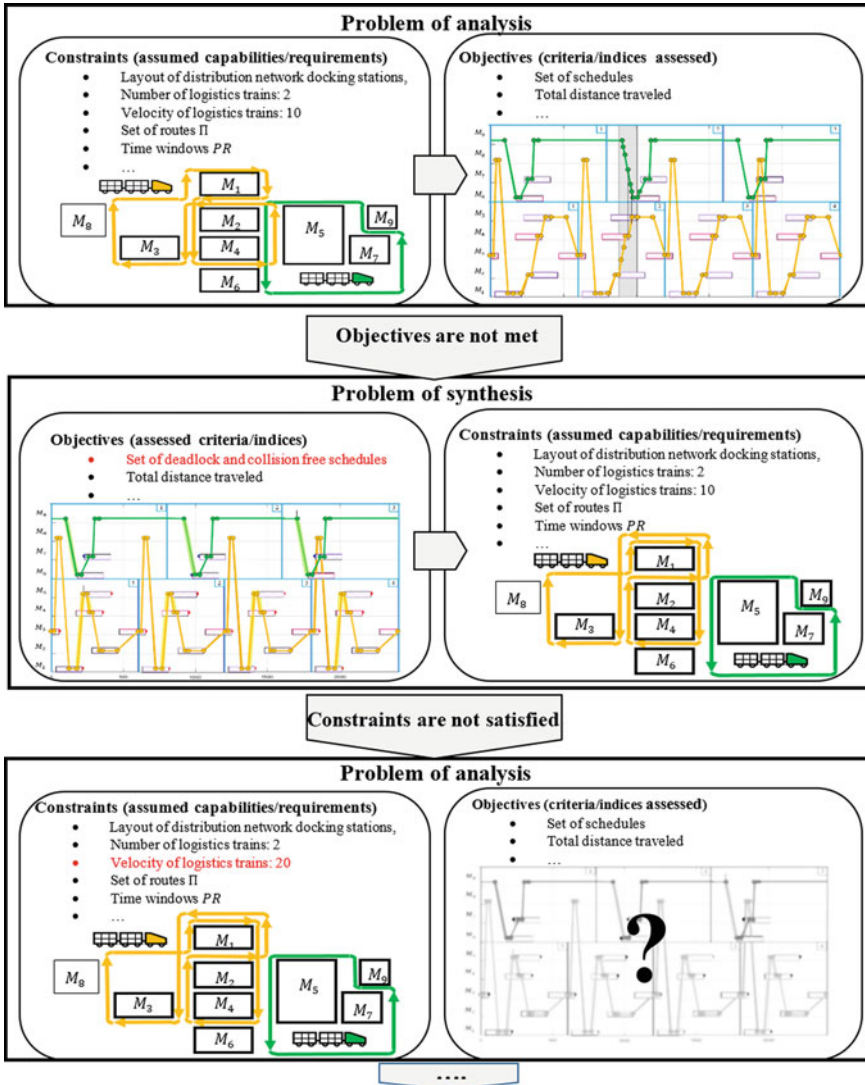


Fig. 7 A flowchart of interactive planning of a milk-run delivery system

## 5 Computational Experiments

Figure 7 shows a flowchart of interactive planning of a milk-run delivery system, the first two stages of which represent, respectively, an analysis and a synthesis problem, are illustrated using the examples discussed in Sect.3.1. (see Figs.4 and 5). The third stage refers to another analysis problem in which trains  $TT_1$  and  $TT_2$ ,

**Table 1** Travel times  $d_{\alpha,\beta}$  between successive docking stations  $M_\alpha$  and  $M_\beta$ 

$d_{\alpha,\beta}$	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$	$M_8$	$M_9$
$M_1$	0	20	45	30	20	40	75	55	25
$M_2$	20	0	70	25	20	25	55	35	30
$M_3$	40	25	0	30	40	30	60	10	50
$M_4$	30	25	30	0	10	10	45	40	30
$M_5$	20	20	40	10	0	20	55	50	20
$M_6$	40	25	30	10	20	0	35	40	40
$M_7$	30	35	60	35	25	35	0	70	5
$M_8$	30	20	10	40	30	40	70	0	40
$M_9$	25	30	55	30	20	40	75	65	0

moving along previously designated routes  $\pi_1 = (M_8, M_1, M_5, M_4, M_2, M_3)$  and  $\pi_2 = (M_9, M_6, M_7)$ , respectively, travel at a double speed. The solution sought at this stage should guarantee cyclic movement of tugger trains (with cycles overlapping with time sub-window periods) in the considered period  $T$ .

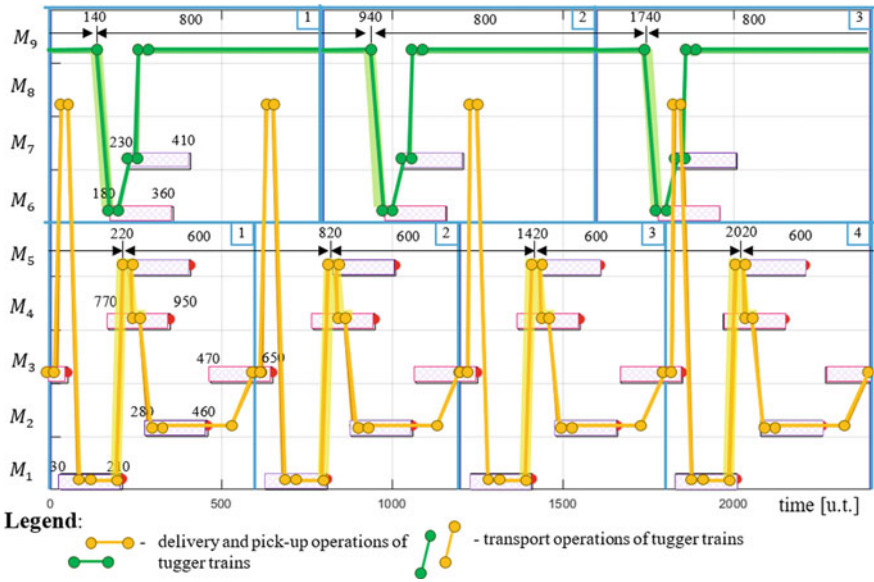
In other words, the goal is to find a cyclic schedule for the tugger trains moving along routes  $\pi_1$  and  $\pi_2$  which would guarantee timely (i.e. within the given time windows) delivery/pick-up of intermediate components/finished products to/from the given docking stations. The solutions sought assume that:

- travel time  $t_{i,j}$  through transportation sector  $(G_i, G_j)$  is twice as short as that shown in Fig. 1.
- travel time  $d_{\alpha,\beta}$  between successive docking stations  $M_\alpha$  and  $M_\beta$  is determined by the shortest path linking these stations—see Table 1.
- the time of each pick-up and delivery operation is the same at  $tr_\alpha = 20$  u.t.
- sequences of time windows  $PR$  are shown in Fig. 1.

The  $CS_l$  problem (18) was implemented and solved for  $l = 4$  time sub-windows in the Oz Mozart constraint programming environment (Windows 10, Intel Core Duo2 3.00 GHz, 4 GB RAM). The first admissible solution, obtained in less than 10 seconds (shown in Fig. 8), provide the final itinerary with deadlock-free and collision-free routings, in which tugger trains move according to an assigned cyclic schedule with a period  $T = LCM(600, 800) = 2400$  u.t.

In contrast to the solution shown in Fig. 5, the solution in Fig. 8 ensures cyclic execution of the transport processes run in a periodically repeating cyclic schedule with period  $T$ —all delivery and pick-up operations are executed cyclically at 600 u.t. and 800 u.t., respectively. However, whether or not this solution can be implemented is conditioned by the possibility of using a fleet of trains that can travel across a shop floor with the given layout twice as fast as in the schedule from Fig. 5.

In addition to the experiments reported above, we compared the effectiveness of the procedures used in the synthesis and analysis problems. The results of the comparison are shown in Table 2. The results of the tests confirm the usefulness of



**Fig. 8** Gantt chart of a deadlock- and collision-free tugger train schedule

**Table 2** Results of computational experiments for the analysis and synthesis problems

Number of		Problem of analysis		Problem of synthesis	
Docking stations	Tugger trains	Does there exist a solution? <sup>a</sup>	Calculation time [s]	Does there exist a solution? <sup>b</sup>	Calculation time [s]
5	1	y	<1	y	<1
5	2	y	<1	y	<1
5	3	n	<1	y	3
5	4	n	<1	y	5
9	1	y	<1	y	5
9 <sup>a</sup>	2	y	< 1	y	10
9	3	n	<1	y	20
9	4	n	<1	y	54
12	1	y	<1	y	35
12	2	n	<1	y	56
12	3	n	<1	y	124
12	4	n	<1	y	202
15	1	y	<1	y	184
15	2	n	<1	y	268
15	3	y	<1	?	>900
15	4	n	<1	?	>900

<sup>a</sup>The solution from Fig. 8

<sup>b</sup>n—there is no solution; y—there exists an admissible solution

the presented solution for fast online prototyping of delivery schedules and transport routes for a fleet of tugger trains. In particular, depending on the adopted assessment criteria, the solver we developed enables interactive search for solutions that minimize the fleet of tugger trains used and allows to assess the possibility of timely execution of planned deliveries for arbitrarily selected tugger train routes. Synthesis problems can be solved online when the number of stops in the system does not exceed fifteen. For systems larger than this, the use of the proposed method leads to combinatorial explosion, which is a natural consequence of the *NP*-hard nature of the problems under consideration.

## 6 Remarks and Conclusions

The novelty of this study is that it presents an integrated modeling approach to milk-run system design and operation. The declarative modeling method used allows one to simultaneously model different, independently processed multi-product production flows as well as the different, independently executed pick-up and delivery processes which service those flows. The model proposed allows a bottom-up or a top-down organization of production flow by integrating the level of workstation-to-workstation transport and the level of flows of batches of different (simultaneously manufactured) products. In the bottom-up approach, the calculated production period determines the routes and cyclic schedules of tugger trains. In the top-down approach, the routes and periods of cyclically moving tugger trains determine the production period. In that context, the model provides a formal framework for formulating task-oriented problems, i.e. analysis and synthesis problems that lie behind a wide range of decisions related to the configuration and/or routing and scheduling of milk-run systems. These decisions concern the selection of a system layout, the size of the tugger train fleet, cyclic schedules for timely pick-up/delivery of materials and so on.

Another benefit of the proposed declarative modeling perspective is that it makes it possible to view the problem under consideration as a constraint satisfaction problem and solve it with the use of constraint programming platforms provided by commercially available software tools, such as CPLEX/ECLiPSe/Gurobi, etc. This means that the investigated problem of congestion-free route planning in in-plant milk-run traffic material supply systems can be implemented and resolved with the help of dedicated, i.e. task-oriented, decision support tools.

The results of the tests demonstrate that the proposed reference model of the recursive constraint satisfaction problem is a useful tool which allows one to formulate the problems of analysis/synthesis of transport routes available in a given distribution system. Implemented computationally, it enables fast online prototyping of supply schedules and transport routes of a fleet of logistics trains. In particular, it allows to interactively search for possibilities of timely execution of planned deliveries. The constraints adopted in the model assume that the concurrent transport processes, executed in a cyclic steady state, have a deterministic nature: resource conflicts leading to deadlocking of processes are resolved using a deadlock prevention method, thus

implementing conditions which guarantee avoidance of congestion. In other words, conditions guaranteeing a cyclic steady state flow of milk-run material delivery processes also ensure their blockage-free execution.

Apart from the research perspective presented in this paper, other directions of study worth mentioning are those aimed at investigating the conditions that would allow one to reschedule milk-runs according to customers' changeable demands. Also interesting is the search for solutions for fuzzy, uncertain decision variables determining supply time windows and robustness of planned routings and schedules, especially in the context of conditions guaranteeing smooth transition between two successive cyclic steady states.

## References

1. Badica, A., Badica, C., Leon, F., Luncean, L.: Declarative representation and solution of vehicle routing with pickup and delivery problem. *Proc. Comput.Sci.* **108**, 958–967 (2017)
2. Bocewicz G., Nielsen P., Banaszak Z., Thibbotuwawa A.: Routing and scheduling of Unmanned Aerial Vehicles subject to cyclic production flow constraints. In: *Proceedings of 15th International Conference on Distributed Computing and Artificial Intelligence* (2018)
3. Bocewicz, G., Nielsen, P., Banaszak, Z.: Declarative modeling of milk-run vehicle routing problem for split and merge supply streams scheduling. *Adv. Intell. Syst. Comput.* **853**, 157–172 (2019)
4. Bocewicz, G., Banaszak, Z., Nielsen, I.: Delivery-flow routing and scheduling subject to constraints imposed by vehicle flows in fractal-like networks. *Arch. Control Sci.* **27**(2), 135–150 (2017)
5. Bocewicz G., Nielsen P., Banaszak Z., Wojcik R.: An analytical modeling approach to cyclic scheduling of multiproduct batch production flows subject to demand and capacity constraints. In: Świątek, J., Borzemski, L., Wilimowska, Z. (eds.) *Information Systems Architecture and Technology: Proceedings of 38th International Conference on Information Systems Architecture and Technology — ISAT 2017*. Springer, Cham (2017)
6. Carić, T., Galić, A., Fosin, J., Gold, H., Reinholz, A.: A Modelling and Optimization Framework for Real-World Vehicle Routing Problems Vehicle Routing Problem. In: Carić, T., Gold, H. (eds.) *Vehicle Routing Problem*, pp. 15–34. I-Tech, Vienna, Austria (2008)
7. Goetschalck, M., Jacobs-Blecha, C.: The vehicle routing problem with backhauls. *Eur. J. Oper. Res.* **42**(1), 39–51 (1989)
8. Gola, A., Kłosowski, G.: Application of fuzzy logic and genetic algorithms in automated works transport organization. In: Omatu, S., Rodríguez, S., Villarrubia, G., Faria, P., Sitek, P., Prieto, J. (eds.) *Distributed Computing and Artificial Intelligence, 14th International Conference DCAI 2017*, pp. 29–36. Springer, Cham (2018)
9. Güner, A.R., Murat, A., Chinnam, R.B.: Dynamic routing for milk-run tours with time windows in stochastic time-dependent networks. *Transp. Res. Part E: Logist. Transp. Rev.* **97**, 251–267 (2017)
10. Gyulaia D., Pfeiffer A., Sobottka T., Vánca J.: Milkrun Vehicle Routing Approach for Shop-floor Logistics. In: *Forty Sixth CIRP Conference on Manufacturing Systems 2013, Procedia CIRP*, vol. 7, pp. 127–132 (2013)
11. Kitamura T., Okamoto K.: Automated route planning for milk-run transport logistics with NuSMV model checker. *IEICE Trans. Inf. Syst.* **E96-D**(12), 2555–2564 (2013)
12. Levner E., Kats V., Alcaide D., Pablo L. Cheng T.C.E: Complexity of cyclic scheduling problems: a state-of-the-art survey. *Comput. Ind. Eng.* **59**(2), 352–361 (2010)

13. Lau, H., Sim, M., Teo, K.: Vehicle routing problem with time windows and a limited number of vehicles. *Eur. J. Oper. Res.* **148**(3), 559–569 (2003)
14. Lenstra J.K., Rinnooy Kan A.H.G.: Complexity of vehicle and scheduling problems. *Networks* **11**(2), 221–227 (1981)
15. Nguyen, P.K., Crainic, T.G., Toulouse, M.: Multi-trip pickup and delivery problem with time windows and synchronization. *Ann. Oper. Res.* **253**(2), 899–934 (2017)
16. Ong, J.O.: Suprayogi: vehicle routing problem with backhaul, multiple trips and time window. *Jurnal Teknik Industri* **13**(1), 1–10 (2011)
17. Patel, D., Patel, M.B., Vadher, J.A.: Implementation of milk run material supply system in vehicle routing problem with simultaneous pickup and delivery. *Int. J. Appl. Innov. Eng. Manag.* **3**(11), 122–124 (2014)
18. Perronnet F., Abbas-Turki A., El Moudni A.: Vehicle routing through deadlock-free policy for cooperative traffic control in a network of intersections: reservation and congestion. In: 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), Qingdao, 8–11 October 2014, pp. 2233–2238. IEEE (2014)
19. Pillac, V., Gendreau, M., Guéret, C., Medaglia, A.L.: A review of dynamic vehicle routing problems. *Eur. J. Oper. Res.* **225**(1), 1–11 (2013)
20. Polak, M., Majdzik, P., Banaszak, Z., Wójcik, R.: The performance evaluation tool for automated prototyping of concurrent cyclic processes. *Fundam. Inf.* **60**(1), 269–289 (2004)
21. Sun S., Gu C., Wan Q., Huang H., Jia X.: CROTPN based collision-free and deadlock-free path planning of AGVs in logistic center. In: Proceedings of the 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), Singapore, 18–21 Nov 2018, pp. 1685–1691. IEEE (2018)
22. Schmidt T., Meinhardt I., Schulze F.: New design guidelines for in-plant milk-run systems <https://pdfs.semanticscholar.org/3fed/4f8d0c253db80c8ae595cd3af494ab120448.pdf> (2016). Accessed 29 Mar 2019
23. Setiani P., Fiddieny H., Setiawan E.B., Cahyanti D.E.: Optimizing delivery route by applying milkrun method. In: Conference on Global Research on Sustainable Transport (GROST 2017). *Advances in Engineering Research*, vol. 147, pp. 748–757 (2017)
24. Sitek P., Wikarek J.: Capacitated vehicle routing problem with pick-up and alternative delivery (CVRPPAD): model and implementation using hybrid approach. *Ann. Oper. Res.* **273**, 257–277 (2001)
25. Suprayogi, Priyandari Y.: Vehicle routing problem with multiple trips, time windows, and simultaneous delivery and pickup services. *Asia Pac. Ind. Eng. Manag. Syst.* **8**, 1543–1552 (2009)
26. Witczak, M., Majdzik, P., Stetter, R., Bocewicz, G.: Interval max-plus fault-tolerant control under resource conflicts and redundancies: application to the seat assembly. *International Journal of Control* (2019), in print
27. Wysk, R.A., Yang, N.-S., Joshi, S.: Resolution of deadlocks in flexible manufacturing systems: avoidance and recovery approaches. *J. Manuf. Syst.* **13**(2), 128–138 (1994)

# **Max-Plus Algebra for Cyclic Systems Modeling**

# Conflict Avoidance Within Max-Plus Fault-Tolerant Control: Application to a Seat Assembly System



Marcin Witczak , Paweł Majdzik , Bogdan Lipiec  and Ralf Stetter 

**Abstract** Flexibility and agility are central requirements for future manufacturing systems (especially assembly systems), because in most industries the product variety and the fluctuations in demand are still increasing. An increase of the degree of flexibility allows more efficient activities aiming at following the dynamically evolving markets. Such systems should be able to react to changes of product, demands, increased varieties of products requirements concerning reduced delivery times and increased product quality. Therefore, a strong focus on the flexibility of manufacturing and assembly systems leads to economic advantages for industrial companies in terms of the system investment cost. In particular, the cost related to the reconfiguration of the system.

## 1 Introduction

In order to fulfill all above requirements, Flexible Manufacturing Systems (FMSs) have to contain typical layers, such as devices layer (industrial robots, conveyor belts, vision systems, sensors etc.), control layer (robot controllers, programmable automation controllers, inverters), and visualization layer (human interface machine).

---

M. Witczak (✉) · P. Majdzik · B. Lipiec  
Institute of Control and Computation Engineering, University of Zielona Góra,  
50 Podgórna Str., 65-246 Zielona Góra, Poland  
e-mail: [m.witczak@issi.uz.zgora.pl](mailto:m.witczak@issi.uz.zgora.pl)

P. Majdzik  
e-mail: [p.majdzik@issi.uz.zgora.pl](mailto:p.majdzik@issi.uz.zgora.pl)

B. Lipiec  
e-mail: [b.lipiec@issi.uz.zgora.pl](mailto:b.lipiec@issi.uz.zgora.pl)

R. Stetter  
Faculty Mechanical Engineering, University of Applied Sciences Ravensburg-Weingarten,  
Weingarten, Germany  
e-mail: [ralf.stetter@hs-weingarten.de](mailto:ralf.stetter@hs-weingarten.de)

© Springer Nature Switzerland AG 2020  
W. Bożejko and G. Bocewicz (eds.), *Modelling and Performance Analysis of Cyclic Systems*, Studies in Systems, Decision and Control 241,  
[https://doi.org/10.1007/978-3-030-27652-2\\_7](https://doi.org/10.1007/978-3-030-27652-2_7)



Manufacturing and assembly systems which employ Automated Guided Vehicles (AGVs) are one important means for enabling flexible operation and agile reconfiguration. However, such systems only allow a smooth and economical operation, if elaborate control and diagnosis systems are present. Today, the Model Predictive Control (MPC) was identified as prominent control concept addressing this challenge, because both the model and the control commands can be continuously updated by using the moving horizon approach. The application of MPC produces two main advantageous aspects. The first advantageous aspect is that the MPC delivers a design procedure for the controller and that it easily can be tuned. The second advantageous aspect is that MPC is able to deal with constraints concerning the inputs and outputs of complex systems. The specific advantages of MPC allow meeting the various precedent requirements regarding manufacturing and assembly processes in industrial companies systems [1–3]. On a certain control level, it is sensible to describe such manufacturing and assembly systems as Discrete Event Systems (DESs) [4]. DESs are event-driven dynamical systems whose state transitions of a DES indicate the physical phenomenon that causes the change in state [5, 6]. One of the primary approaches to evaluate the performance of FMSs is the simulation. The most important advantages of this approach are that it can be used for arbitrary classes of DESs, however this approach requires tedious simulation runs and cannot provide an understanding of the dependence of parameters. The other approach allows calculating and analyzing the system performance using an algebraic model, e.g. max-plus algebra model.

This chapter illustrated a novel control scheme that is based on the general idea to apply the max-plus algebra for MPC (compare [7]). Up to now this idea only allowed to describe DESs without resource conflicts. The novel control scheme presented in this chapter also allows to control DES with such conflicts. It is another advantageous aspect that the novel scheme includes a representation of uncertain discrete event systems which are influenced by internal and unobservable events (compare [8]). Additionally, the scheme includes an active fault-tolerant control framework which allows to identify faults and to accommodate these faults accordingly [9]. It is important to note that the application of this kind of elaborate control and diagnosis scheme can be enabled and eased if the control and diagnosis system is consciously designed. Guidelines for this kind of design can be summarized under the notion “Design for Control”; the next section is focusing on this topic.

The most important features related to the design for control are presented in Sect. 2. In Sect. 3 an overview of the assembly system is given. The modelling of the assembly system is described in Sect. 4. Section 5 presents modelling of AGVs, and in Sect. 5.2 describe predictive control of two AGVs. Section 6 is devoted to fault tolerant control.

## 2 Design for Control

In recent years, research projects were initiated which aim at the development of design guidelines which aim at support the development of technical systems that enable and ease control. These guidelines were summarized under the notion “Design for Control (DfC)”. The term “Control” includes a large number of different activities with the aim to manage, command, direct or regulate the behaviour of technical systems. One example for a DfC guideline is the recommended use of over-actuation, i.e. the application of more or stronger actuators than directly necessary [10]. It is rather obvious that the possibility for control actions is enhanced by means of this kind of over-actuation. The focus of this chapter is a complex system with many AGVs. Today, complex systems are usually realized with modules which reduce the complexity and allow reuse. One insight concerning the DfC is that structures should be congruent, i.e. for modular technical system also the control system should be modular [10]. Another important insight is that modules should contain local intelligence for local control loops [10]. For the operation of AGVs it is sensible to distribute control and diagnosis tasks to the individual AGVs and even their components in order to optimize control and diagnosis speed and to avoid excessive requirements for the communication between AGVs. Additionally, in order to reduce complexity, it is sensible to realize planning, control and diagnosis systems with certain hierarchies (compare [11]). Figure 1 shows a hierarchical and distributed control and diagnosis concept with an FTC-MPC layer.

Meanwhile it is an established fact that control and diagnosis systems will only be applied in industrial companies, if they are an integral part of the production system

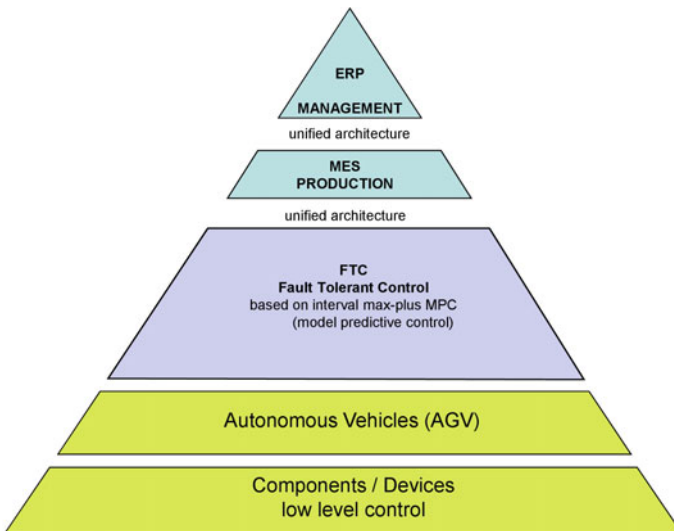


Fig. 1 Hierarchical and distributed control and diagnosis concept

information infrastructure (compare [11]), namely the Enterprise Resource Planning (ERP) system and Manufacturing Execution System (MES). Enterprise resource planning (ERP) is an integrated computer-based system used to manage internal and external resources including tangible assets, financial resources, materials, and human resources. On the next level below are Manufacturing Execution Systems (MES). This level takes this planning output of the ERP and executes this plan in the production. A fault-tolerant control system, as proposed in this chapter, needs to communicate with the MES.

The rapid development of information technology of the last decades enables to intensify the collection and processing of all kinds of data and information in production systems. The culmination of this data collection and processing is the so-called “Digital Twin” of the production system. A digital twin can be defined in the following manner [12]: a digital twin is an integrated multi-physics, multi-scale, probabilistic simulation of a complex technical system which employs the best available physical models, sensor readings, sensor information updates, etc., to mirror the life of its corresponding twin—the real technical system. A digital twin consists of three parts [13]:

- the original technical system in real space,
- a digital product in a virtual space and
- the connection of data and information which links the two spaces.

Digital twins are virtual images of physical objects or systems. Digital Twins of manufacturing and assembly systems significantly contribute to the required transparency and to near real-time production control [14]. The compulsory precursor of the digital twin is the Internet of Things (IoT). Digital Twins dispose of four essential entities:

- sensors which allow a detailed, far reaching monitoring of current status
- connectivity, which realizes a networks between the modules of the systems
- defined data structures enabling analytic functionalities
- a user interface that visualizes the relevant data and information

Examples for realized digital twins are digital twin driven product manufacturing in shop floors and product services [15]. The Digital Twins concept and its additional digital functions enable the monitoring and control of real counterparts—real technical systems. In addition, digital twins communicate with each other and with higher architectural levels. For the AGV system under consideration, this general concept is shown in Fig. 2.

In Fig. 2 it is visible that the hierarchical and distributed control structure of the original technical system is represented in the digital twin. A continuous update is necessary at all levels. Between the levels information flows are present such as sensor readings and assignments (here the term “assignments” is a general term for information or commands such as required schedules or arrival times at certain points in spaces). These information flows also have to be represented in the digital twin and require continuous update.

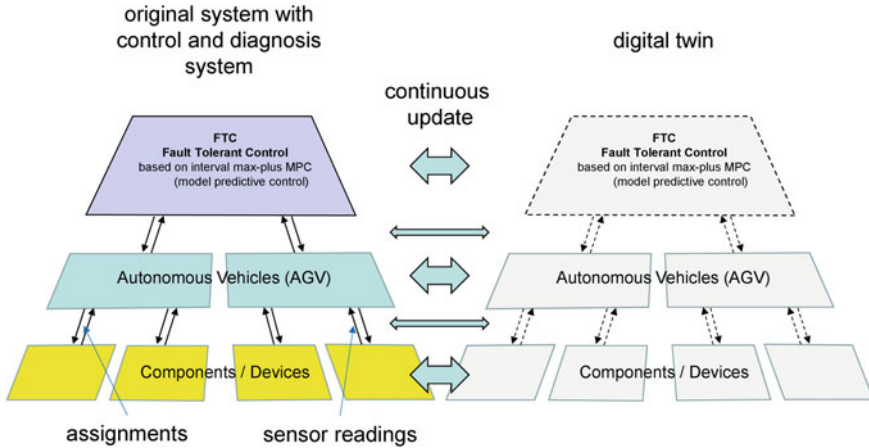


Fig. 2 Digital twin

This chapter concentrates on a fault-tolerant control framework that is located on the highest level of Fig. 2. In the next sections the exemplary system is illustrated and the modelling possibilities for this level are explained.

### 3 Overview of the Assembly System

The considered manufacturing system consists of two main parts (see Fig. 3). The first part constitutes an assembly system that produces the car’s seats. The second one is a transportation system that transmit the seats from the assembly system to the high storage warehouse. One of the most flexible transport means for in-plant transportation are AGVs. AGVs dispose of further advantageous characteristics such as comparatively low investment costs and relatively small expenditures for elements of the plant infrastructure. The objective of this section is to describe the individual production tasks in the assembly system (see Fig. 4). The assembly system can be considered as the system belonging to the class of DES. The entire description of DES should contain the following parameters:

- $k$  event counter;
- $R_i$   $i$ th processing unit;
- $d_i$   $i$ th processing time;
- $u_i(k)$  time instant at which the product is transferred to the system’s  $i$ th input;
- $x_i(k)$  time instant at which  $i$ th processing unit starts to carry out a demanded task;
- $y_i(k)$  time of delivering the  $i$ th product;
- $t_{i,j}$  time for transportation from the  $i$ th to the  $j$ th processing unit;
- $t_{in,i}$  time for transportation from the  $i$ th input storage to the  $i$ th processing unit.

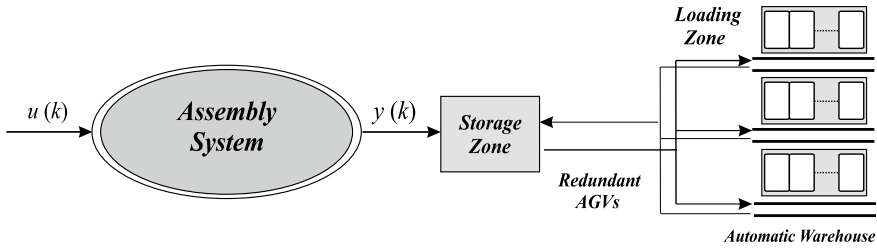
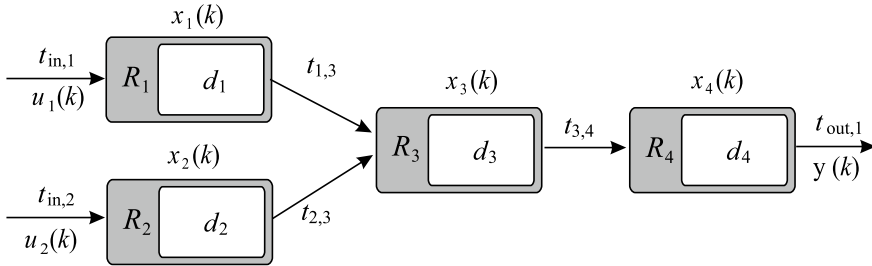


Fig. 3 Overview of the manufacturing system



- $R_1$  - assembly station - lower seat frame,  $R_2$  - assembly station - back rest seat,
- $R_3$  - assembly station - connection frame and back rest,
- $R_4$  - station - assessment of quality,

Fig. 4 Details of the assembly system

Experts in automotive industry expect a profound change of the use of cars in the next decade. The next levels of autonomous driving will enable drivers and passengers to use the space in their cars in a completely different manner. This influences nearly all components of the car interior and especially the seats. Future seat will need to dispose of integrated safety systems such as seat integrated belts and airbags. Additionally, even more comfort features such as climate control, massage functions and personal audio will be integrated in the seats. This will lead to heavier seats which require more space. This will also lead to changes in factory transportation systems. One possibility to address the difficult transportation tasks of future seats are AGVs, which are flexible enough for a large product variety and fluctuations of demand. A prospective seating assembly systems is shown in Fig. 4. The process starts with two parallel assembly stations—one for the assembly of the lower part of the seat (resource  $R_1$  with processing time  $d_1$ ) and one for the assembly of the back rest (resource  $R_2$  with processing time  $d_2$ ). Both parts are then delivered to a common assembly station which connects the two parts (resource  $R_3$  with processing time  $d_3$ ). Subsequently, seats are transported to the station 4 ( $R_4$  with processing time  $d_4$ ), where the quality of final products are checked. The finished seats are then

transported to the storage zone. From this station, several AGVs transport the seats to the loading zone (i.e. the automatic warehouse).

Having the precedent formal description of the system and the mathematical background that is introduced in next section, it is possible to determine the mathematical model of the assembly system and the redundant AGVs.

## 4 Modelling of the Assembly System

This section explains the main mathematical concepts describing the max-plus algebra formalism and to present the max-plus algebra linear space equation of the assembly system (described in the previous section, Fig. 4).

### 4.1 Max-Plus Algebra Formalism

It is possible to define the basic structure of the so-called max-plus algebra  $(\mathbb{R}_{max}, \oplus, \otimes)$  as formulated subsequently:

$$\begin{aligned} \mathbb{R}_{max} &\triangleq \mathbb{R} \cup \{-\infty\}, \\ \forall a, b \in \mathbb{R}_{max}, a \oplus b &= \max(a, b), \\ \forall a, b \in \mathbb{R}_{max}, a \otimes b &= a + b, \end{aligned} \tag{1}$$

where  $\mathbb{R}_{max}$  is the field of real numbers.

The first operator  $\oplus$  describes the max-plus algebraic addition while the second operator  $\otimes$  stands for the max-plus algebraic multiplication.

The fundamental characteristics of these max-plus algebra operators may be formulated in the subsequent form:

$$\begin{aligned} \forall a \in \mathbb{R}_{max} : a \oplus \varepsilon &= a \text{ and } a \otimes \varepsilon = \varepsilon, \\ \forall a \in \mathbb{R}_{max} : a \otimes e &= a, \end{aligned} \tag{2}$$

In these equations  $\varepsilon = -\infty$  and  $e = 0$  act as neutral elements for both the max-plus algebraic addition and for the max-plus algebraic multiplication operators.

It is important to note that the max-plus algebra operations are associative, commutative and distributive in the same manner as in conventional algebra. Thus, the subsequent properties can be formulated:

$$\begin{aligned}
&\text{associativity of addition} && \forall a, b, c \in \mathbb{R}_{max} : (a \oplus b) \oplus c = a \oplus (b \oplus c), \\
&\text{commutativity of addition} && \forall a, b \in \mathbb{R}_{max} : (a \oplus b) = b \oplus a, \\
&\text{associativity of multiplication} && \forall a, b, c \in \mathbb{R}_{max} : (a \otimes b) \otimes c = a \otimes (b \otimes c)
\end{aligned} \tag{3}$$

Two important aspects of max-plus algebra are that it does not have additive inverses and it is idempotent. This is why max-plus algebra is considered a semiring and not a ring. For matrices  $\mathbf{X}, \mathbf{Y} \in \mathbb{R}_{max}^{m \times n}$  and  $\mathbf{Z} \in \mathbb{R}_{max}^{n \times p}$

$$(\mathbf{X} \oplus \mathbf{Y})_{ij} = x_{ij} \oplus y_{ij} = \max(x_{ij}, y_{ij}), \tag{4}$$

$$(\mathbf{X} \otimes \mathbf{Z})_{ij} = \bigoplus_{k=1}^n x_{ik} \otimes z_{kj} = \max_{k=1, \dots, n} (x_{ik} + z_{kj}). \tag{5}$$

The publications [16, 17] contain further details and definitions concerning the formalism of max-plus algebra.

## 4.2 Max-Plus Linear System

One of the challenges of the work with DESs has its origin in the fact that DESs necessitate a non-linear description, if they are modelled in conventional algebra. Nevertheless, it was possible in recent years to find a specific class of DES that are named max-plus linear systems. Linear max-plus models only enable the synchronization of tasks but do not allow an occurrence of concurrency. Consequently, DESs can be modelled in the subsequent form employing the max-plus algebra formalism:

$$\mathbf{x}(k+1) = \mathbf{A} \otimes \mathbf{x}(k) \oplus \mathbf{B} \otimes \mathbf{u}(k+1), \tag{6}$$

$$\mathbf{y}(k) = \mathbf{C} \otimes \mathbf{x}(k), \tag{7}$$

the index  $k$  serves as event counter and:

- $\mathbf{x}(k) \in \mathbb{R}_{max}^n$  designates the state, which contains the time instants corresponding to the internal events occurring at  $k$ ,
- $\mathbf{u}(k) \in \mathbb{R}_{max}^r$  designates the input vector, which contains the time instants corresponding to input events occurring at  $k$ ,
- $\mathbf{y}(k) \in \mathbb{R}_{max}^m$  designates the output vector, which contains the time instants corresponding to the output events occurring at  $k$ ,
- $\mathbf{A} \in \mathbb{R}_{max}^{n \times n}$  designates the state transition matrix,  $\mathbf{B} \in \mathbb{R}_{max}^{n \times r}$  designates the control matrix and  $\mathbf{C} \in \mathbb{R}_{max}^{m \times n}$  designates the output matrix.

The basic challenge in the development of described assembly system is to design and implement an appropriate synchronization rules for all tasks, both processing and transportation tasks. Generally, two essential synchronization modes, i.e. a mutual

exclusion mode and a rendez-vous mode, can be distinguished. The mutual exclusion mode requires that at the same time only one task can perform its operation on the shared resource. The rendez-vous mode involves the case where two or more tasks have to finish its operations so that the next operation can start its performance.

In the system described above, two of the modes of synchronization rules, which were mentioned earlier in this chapter, are present. The first mode of synchronization rules describes the phenomenon that any processing unit may start performing its intended operation on a next product (in the  $k + 1$ th iteration) as soon as the earlier processing operations on the previous product have been successfully carried out (in the  $k$ th iteration). This mode of synchronization concerning the  $R_1$  unit (see Fig. 4) can be expressed by the subsequent equation:

$$x_1(k + 1) = \max(x_1(k) + d_1, u_1(k + 1) + t_{in,1}) \quad (8)$$

It is obvious that this kind of synchronization has to be applied for each assembly station in the system (9).

The second mode of synchronization represents the rendez-vous mode and concerns the unit  $R_3$  (see Fig. 4) that is used by tasks from two assembly cycles. Taking into account the structure of described system, the operations on  $R_1$  and  $R_2$  have to be finished in order to the assembly operation on unit  $R_3$  can be started. This mode of synchronization is represented by:

$$x_3(k + 1) = \max(x_1(k + 1) + d_1 + t_{1,3}, x_2(k + 1) + d_2 + t_{2,3}, x_3(k) + d_3)$$

In one takes the preceding assumptions as well as the modes of synchronization into consideration, it is possible to describe the system from Fig. 4 using the following model:

$$\begin{aligned} x_1(k + 1) &= \max(x_1(k) + d_1, u_1(k + 1) + t_{in,1}) \\ x_2(k + 1) &= \max(x_2(k) + d_2, u_2(k + 1) + t_{in,2}) \\ x_3(k + 1) &= \max(x_1(k + 1) + d_1 + t_{1,3}, x_2(k + 1) + d_2 + t_{2,3}, x_3(k) + d_3) = \\ &\quad \max(x_1(k) + 2d_1 + t_{1,3}, x_2(k) + 2d_2 + t_{2,3}, x_3(k) + d_3, \\ &\quad u_1(k + 1) + d_1 + t_{in,1} + t_{1,3}, u_2(k + 1) + d_2 + t_{in,2} + t_{2,3}) \\ x_4(k + 1) &= \max(x_3(k + 1) + d_3 + t_{3,4}, x_4(k + 1) + d_4) = \\ &\quad \max(x_1(k) + 2d_1 + d_3 + t_{1,3} + t_{3,4}, x_2(k) + 2d_2 + d_3 + t_{2,3} + t_{3,4}, \\ &\quad x_3(k) + 2d_3 + t_{3,4}, x_4(k + 1) + d_4), u_1(k + 1) + d_1 + d_3 + t_{in,1} + \\ &\quad t_{1,3} + t_{3,4}, u_2(k + 1) + d_2 + d_3 + t_{in,2} + t_{2,3} + t_{3,4}) \quad (9) \\ \bar{y}(k) &= x_4(k) + d_4 + t_{out,1} \end{aligned}$$

One may also describe the equations listed above using a compact form (6)–(7) while a detailed description of the system matrices is given in (10).



$$\begin{aligned}
A &= \begin{bmatrix} d_1 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & d_2 & \varepsilon & \varepsilon \\ 2d_1 + t_{1,3} & 2d_2 + t_{2,3} & d_3 & \varepsilon \\ 2d_1 + d_3 + t_{1,3} + t_{3,4} & 2d_2 + d_3 + t_{2,3} + t_{3,4} & 2d_3 + t_{3,4} & d_4 \end{bmatrix}, \\
B &= \begin{bmatrix} t_{in,1} & \varepsilon \\ \varepsilon & t_{in,2} \\ d_1 + t_{in,1} + t_{1,3} & d_2 + t_{in,2} + t_{2,3} \\ d_1 + d_3 + t_{in,1} + t_{1,3} + t_{3,4} & d_2 + d_3 + t_{in,2} + t_{2,3} + t_{3,4} \end{bmatrix}, \\
C &= [\varepsilon \ \varepsilon \ \varepsilon \ d_4 + t_{out,1}].
\end{aligned} \tag{10}$$

In consideration of the fact that an analytical description of the system is present, the processing and transportation times can be incorporated within an analytical description (Eq. (11)), which are:  $d_1 = 1$ ,  $d_2 = 2$ ,  $d_3 = 2$ ,  $d_4 = 1$ ,  $t_{1,3} = 4$ ,  $t_{2,3} = 1$ ,  $t_{3,4} = 2$ ,  $t_{in,1} = 2$ ,  $t_{in,2} = 1$ ,  $t_{out,1} = 2$ .

$$A = \begin{bmatrix} 1 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 2 & \varepsilon & \varepsilon \\ 6 & 5 & \varepsilon & \varepsilon \\ 10 & 9 & 6 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & \varepsilon \\ \varepsilon & 1 \\ 7 & \varepsilon \\ 11 & 8 \end{bmatrix}, \quad C = [\varepsilon \ \varepsilon \ \varepsilon \ 4] \tag{11}$$

### 4.3 Handling Constraints

For the sake of describing the full functionality of the system, it is inevitable to generate a set of constraints which limit system behavior. The constraints of the system can be described in the subsequent form:

- The first constraint describes the fact that the system has to follow a predefined trajectory. It is possible to define this trajectory by employing scheduling constraints of the subsequent form:

$$t_{ref,j}(k) \geq x_j(k), \quad j = 1, \dots, n. \tag{12}$$

In this expression,  $t_{ref,j}(k)$  stands for the upper bound of  $x_j(k)$  at the time instant  $k$ .

- The second constraint is directly linked to the second mode of synchronization. It facilitates the avoiding of tasks which are waiting (see Sect. 4.2):

$$\forall i \in \{1, 2, \dots, n\} \quad ((x_i(k+1) - (x_i(k) + d_i)) \leq 0), \tag{13}$$

In this expression,  $n$  denotes the size of system; this size is equal to the number of present processing units.

- The third constraint is directly linked to the performance of the AGV,

$$\bar{u}_i \geq u_i(k+1) - u_i(k), \quad i = 1, \dots, r. \tag{14}$$

It is important to note that the upper bound  $\bar{u}_i$  stands for the maximum velocity the AGV may achieve. A crossing of this limit may lead to a dramatic increase of the energy consumption of the drives of the AGV.

- The fourth and last constraint is concerning the change rate:

$$u_j(k+1) - u_j(k) \geq z_j, \quad j = 1, \dots, r. \quad (15)$$

In this expression,  $z_j > 0$  designates the upper bound of the change rate.

One additional obvious constraint is the fact that the time to reach any individual assembly station for  $k+1$  needs to be larger than or at least equal to the one for  $k$ .

#### 4.4 Constrained Model Predictive Control

Current industrial production systems require constraints and certain control quality measures. One central advantage of MPC is its natural ability of dealing with constraints, therefore it is an ideal candidate to address the challenges of current industrial production systems. The framework, which is proposed in this chapter, could be developed on the basis of the general MPC strategy for max-plus linear systems described in [7]. In the proposed scheme, MPC and max-plus algebra are applied in order to reduce the number of conflict tasks. The core of the problem is to find the input sequence  $u(k), \dots, u(k+N_p-1)$  minimizing the cost function  $J(u)$

$$J(u) = - \sum_{j=0}^{N_p-1} \sum_{i=1}^r q_i u_i(k+j), \quad (16)$$

where  $q_i > 0$ ,  $i = 1, \dots, m$  denotes a positive weighting constant, while  $N_p$  designates the prediction horizon. It is a core advantage of (16) compared to the quadratic criteria employed in the case of continuous systems that no time-consuming quadratic programming is required. On the contrary, an efficient linear programming framework may be applied, because of the linear constraints (12)–(15).

The first inevitable step that leads to a possible computational framework is to make sure that no direct influence of  $\mathbf{x}(k+1), \dots, \mathbf{x}(k+N_p-1)$  to the scheduling constraints (12) exists. In order to achieve this, let:

$$\tilde{\mathbf{x}}(k+N_p-1) = \mathbf{M} \otimes \mathbf{x}(k) \oplus \mathbf{H} \otimes \tilde{\mathbf{u}}(k), \quad (17)$$

where

$$\tilde{\mathbf{u}}(k) = \begin{bmatrix} \mathbf{u}(k+1) \\ \mathbf{u}(k+2) \\ \vdots \\ \mathbf{u}(k+N_p-1) \end{bmatrix}, \quad \tilde{\mathbf{x}}(k+N_p-1) = \begin{bmatrix} \mathbf{x}(k+1) \\ \vdots \\ \mathbf{x}(k+N_p-1) \end{bmatrix}. \quad (18)$$

On the basis of the description of the DES formulated in (6)–(7), it may be shown that:

$$\mathbf{H} = \begin{bmatrix} \mathbf{B} & \varepsilon & \cdots & \varepsilon \\ \mathbf{A} \otimes \mathbf{B} & \mathbf{B} & \cdots & \varepsilon \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^{\otimes N_p-2} \otimes \mathbf{B} & \mathbf{A}^{\otimes N_p-3} \otimes \mathbf{B} & \cdots & \mathbf{B} \end{bmatrix}, \quad \mathbf{M} = \begin{bmatrix} \mathbf{A} \\ \mathbf{A}^{\otimes 2} \\ \vdots \\ \mathbf{A}^{\otimes N_p-1} \end{bmatrix}.$$

It is possible to formulate the intended optimization strategy in a straight-forward manner. An initial condition  $x(k)$  needs to be determined. Starting from this condition, the optimal input sequence  $\tilde{\mathbf{u}}(k)^*$  may be found by means of solving:

$$\tilde{\mathbf{u}}(k)^* = \arg \min_{\tilde{\mathbf{u}}(k)} J(\mathbf{u}), \quad (19)$$

considering the constraints (12)–(15).

All associated constraints need to be provided, before the scheme can be applied to the assembly system (Fig. 2). The first logical step are the scheduling constraints:

$$\begin{aligned} t_{ref}(0) &= [1, 2, 7, 11]^T, \\ t_{ref}(1) &= [3, 3, 8, 12]^T, \\ &\vdots \end{aligned} \quad (20)$$

The prediction horizon was set to  $N_p = 4$  along with  $q_1 = q_2 = 1$  shaping the cost function (16). The goal of this example is to show the performance of the scheme in case of a chosen schedule ( $t_{ref}$ ) under a resource conflict. As can be observed in Fig. 6, a conflict on  $R_3$  arises for  $k = 4$ . The proposed scheme along with suitable constraints allows an appropriate control of the process tasks by optimized acceleration/deceleration of the pre-product providing time to  $R_1$  from  $u_1$  and/or to  $R_2$  from  $u_2$ . While analyzing Fig. 5, it can be observed that an appropriate control of  $u_1$  at  $k = 3$  allows avoiding the conflict described above.

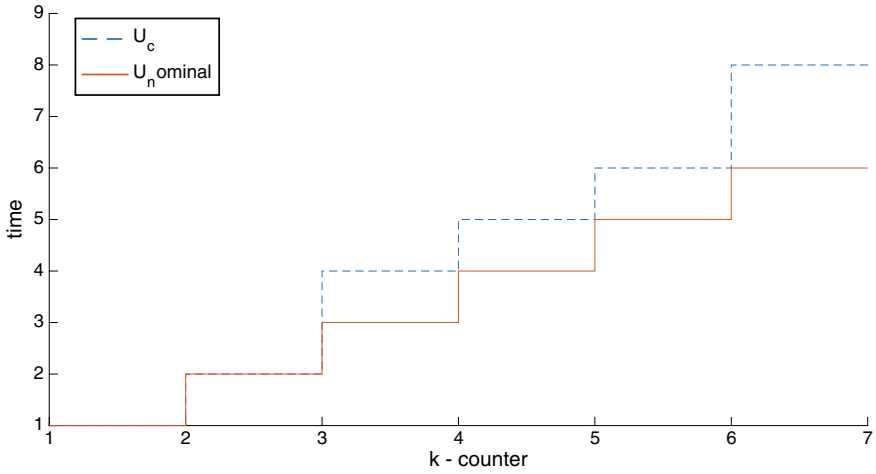


Fig. 5 Evolution of control variable with proposed strategy (dashed line) and without it (solid line)

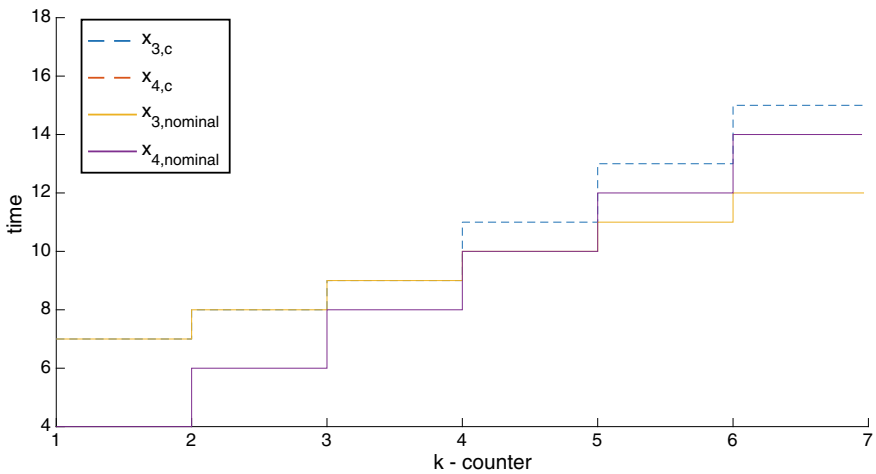


Fig. 6 The states  $x_3(k)$  and  $x_4(k)$  with MPC (dashed line) and without it (solid line)

### 5 Modelling of the AGVs

As it was described in Sect. 3, the overall system consists of two parts, where the second one constitutes the transportation system that is based on AGVs. AGVs are responsible for delivering given final products (seats) from the assembly outlet towards appropriate point of the warehouse. The warehouse has high-rise shelves on which pallets with products are stored. Between the shelves are aisles for automated

forklifts. The advantages of such high storage warehouse are: good access to articles, economical use of space and pressure-avoiding storing of the goods ([18]).

The unique design of the AGVs allows unlimited manoeuvring possibilities (see [19] for a comprehensive explanation). AGVs system ensures the high flexibility and relatively large fault-tolerance. They can theoretically drive in the zone in front of the warehouse and can supply and receive products on palettes to and from dedicated transfer stations. The feeding system consists of three control levels with a hierarchical control structure. The lowest level controls the continuous base-line including physical and virtual sensors. An middle control level is applied for detailed path planning. The highest control level called “supervisory control level” is responsible for dispatching AGVs and for controlling transportation times. This supervisory control level is in the core subject of the research described in this section.

Because of safety requirements, AGVs have to move along a designed lanes which are intended to forward and backward movement.

$$\mathbb{M}(k) = [c(k), b(k), d(k), p(k)], \quad (21)$$

where:

- $c(k)$  denotes the item packing and transportation time from the outlet of the production system to  $p(k)$  transfer station;
- $b(k)$  denotes the item unpacking and transportation time from  $p(k)$  transfer station to the production outlet;
- $d(k)$  is the minimum acceptable time difference between delivering  $k - 1$ th item to  $p(k - 1)$  transfer station and  $k$ th item to  $p(k)$  transfer station, respectively;
- $p(k)$  is a unique number identifying the transfer station, i.e.,  $p(k) \in \{1, \dots, n_s\}$  where  $n_s$  is the number of transfer stations.

Moreover, the sequence of the items which have to be transported from the production outlet to the transfer stations are supplied by MES:

$$\mathbb{M}(0), \mathbb{M}(1), \dots, \mathbb{M}(N_p - 1), \quad (22)$$

where  $N_p$  stands for production horizon. It should be noted that each  $k$ th item have to be delivered to the  $p(k)$  transfer station according to an assumed time schedule:

$$x_{ref}(0), x_{ref}(1), \dots, x_{ref}(N_p - 1), \quad (23)$$

In order to achieve this aim, the schedule of  $n_v$  AGVs has to be dispatched with along a sequence of item outlet delivery times:

$$y(0), y(1), \dots, y(N_p - 1). \quad (24)$$

These delivery times stand for the time of providing the  $k$ th item at the outlet of the production system. In this chapter, the performance of the AGV-based transportation system is measured in the following form:

$$J(y) = - \sum_{k=0}^{N_p-1} y(k). \quad (25)$$

This preceding function needs to be minimized taking into consideration the scheduling constraint (23) while also considering the overall performance of the AGVs. Resulting from this, the largest possible sum of (24) needs to be obtained, which guarantees the satisfaction of (23). It is important to note that (25) may also be defined in a different fashion, e.g. by means of allowing the maximization of the consecutive differences  $y(k+1) - y(k)$ . This arrangement may provide the maximum spread between consecutive item outlet delivery times. For the sake of simplicity and clarity, this chapter concentrates on an transportation system consisting of two AGVs.

An mathematical description of two AGVs has to be defined employing an extended max-plus algebra which can be based on the max-plus algebra presented in Sect. 4. Additionally, for obtaining a sequence (24) which maximizes (25) taking into consideration the scheduling constraint (23), the model predictive control employing the max-plus algebra description is used. The preceding approach assumes that the actual transportation times of the  $i$ th AGV, which carries the  $k$ th item, are equal to their nominal values, even though the second AGV transportation times are set to zero, i.e.:

$$\text{if } c_1(k) = c(k), b_1(k) = b(k) \text{ then } c_2(k) = 0, b_2(k) = 0 \quad (26)$$

$$\text{if } c_2(k) = c(k), b_2(k) = b(k) \text{ then } c_1(k) = 0, b_1(k) = 0 \quad (27)$$

Transportation delays for which the actual measured transportation times  $c_i(k)^m$  and  $b_i(k)^m$  are not equal  $c(k)$  and  $b(k)$  respectively, are considered to be *faults*. This process may be formally described as:

$$\begin{aligned} \text{if } c_i(k)^m = c(k) \text{ then } f_{i,c}(k) &= 0 \\ \text{else } f_{i,c}(k) &= c_i(k)^m - c(k) \end{aligned} \quad (28)$$

$$\begin{aligned} \text{if } b_i(k)^m = b(k) \text{ then } f_{i,b}(k) &= 0 \\ \text{else } f_{i,b}(k) &= b_i(k)^m - b(k) \quad i \in \{1, 2\} \end{aligned} \quad (29)$$

## 5.1 Mathematical Description of AGVs

This section aims to deliver a mathematical description which will enable the fault-tolerant control of the AGV system. The core of this section is a mathematical description of twin AGVs which allows a real-time determination of their time schedule on a given horizon  $N_p$ . The initial step can be a definition of the main variables:

$x_i(k)$  denotes the time instant at which the  $i$ th AGV is ready to transport the  $k$ th item,  $i \in \{1, 2\}$ ;

$x_3(k)$  denotes  $k$ th item delivery time at the  $p(k)$  transfer station;  
 $v_i(k)$  denotes decision variable that associates  $i$ th AGV with the transportation of  $k$ th item;  $v_i(k) \in \{e, \varepsilon\}$ ,  $i \in \{1, 2\}$ .

Note that  $v_i(k) = e$  means that the  $i$ th AVG transports the  $k$ th item while  $v_i(k) = \varepsilon$  means an opposite situation. On the basis of the variables, which were defined precedently, the time-evolution of  $x_i(k)$  for each AGV can be described in the subsequent form:

$$\begin{aligned} x_1(k) &= \max(x_1(k-1) + b_1(k-1) + c_1(k-1), y(k) + v_1(k)), \\ x_2(k) &= \max(x_2(k-1) + b_2(k-1) + c_2(k-1), y(k) + v_2(k)). \end{aligned} \quad (30)$$

with the associated constraints

$$\begin{aligned} b_1(k) &= \max(e, b(k) + v_1(k)), \\ b_2(k) &= \max(e, b(k) + v_2(k)), \\ c_1(k) &= \max(e, c(k) + v_1(k)), \\ c_2(k) &= \max(e, c(k) + v_2(k)). \end{aligned} \quad (31)$$

and

$$\begin{aligned} v_1(k) = e &\Leftrightarrow v_2(k) = \varepsilon \\ v_2(k) = e &\Leftrightarrow v_1(k) = \varepsilon \end{aligned} \quad (32)$$

Note that from (42) follows that only one AGV, i.e.  $i$ th AGV can transport  $k$ th item from the production outlet towards  $p(k)$ th transfer station. Subsequently, the  $k$ th item delivery time at  $p(k)$ th transfer station obeys:

$$x_3(k) = \max(x_1(k) + c_1(k) + v_1(k), x_2(k) + c_2(k) + v_2(k), x_3(k) + d_3(k)) \quad (33)$$

On the basis of (31) it is possible to show that

$$\begin{aligned} c_1(k) + v_1(k) &= \max(e, c(k) + v_1(k)) + v_1(k) = c(k) + v_1(k) \\ c_2(k) + v_2(k) &= \max(e, c(k) + v_2(k)) + v_2(k) = c(k) + v_2(k) \end{aligned} \quad (34)$$

and for this reason (35) can be condensed to:

$$x_3(k) = \max(x_1(k) + c(k) + v_1(k), x_2(k) + c(k) + v_2(k), x_3(k-1) + d_3(k)) \quad (35)$$

Through a detailed analysis of (30)–(35), it becomes obvious that the only employed mathematical operators are  $+$  and  $\max$ . On that account, among the different available DES modelling techniques [20–22], the max plus algebra [7, 23] is apparently

the most suitable one. Employing the preceding notation, it may be proposed to reformulate the model (30)–(35) into the subsequent form:

$$x(k) = A(v(k-1), v(k), k) \otimes x(k-1) \oplus B(v(k), k) \otimes y(k), \quad (36)$$

where:  $\mathbf{x}(k) = [x_1(k), x_2(k), x_3(k)]^T$ ,  $v(k) = [v_1(k), v_2(k)]$ ,  $\mathbf{A}(v(k-1), v(k), k) \in \mathbb{R}_{max}^{n \times n}$  and  $\mathbf{B}(v(k), k) \in \mathbb{R}_{max}^{n \times r}$  stand for the state transition matrix and the control matrix, respectively.

For a convenient application (and with a slight abuse of the notation conventions) the above matrices will be denoted by  $A_v(k)$  and  $B_v(k)$ . On that account, substituting (30) into (35) leads to:

$$\begin{aligned} x_3(k) &= \max(x_1(k-1) + b_1(k-1) + c_1(k-1) + c_1(k) + v_1(k), \\ & x_2(k-1) + b_2(k-1) + c_2(k-1) + c_2(k) + v_2(k), \\ & y(k) + c(k) + v_1(k), y(k) + c(k) + v_2(k), \\ & y(k) + c(k) + v_3(k), \dots, y(k) + c(k), \\ & x_3(k-1) + d(k)) \end{aligned} \quad (37)$$

Combining (30) and (37) makes it possible to derive the matrices  $A_v(k)$  and  $B_v(k)$  that are given by (38)

$$\begin{aligned} A_v(k) &= \begin{bmatrix} b_1(k-1) + c_1(k-1) & \varepsilon & \varepsilon \\ \varepsilon & b_2(k-1) + c_2(k-1) & \varepsilon \\ b_1(k-1) + c_1(k-1) + c_1(k) + v_1(k) & b_2(k-1) + c_2(k-1) + c_2(k) + v_2(k) & d_3(k) \end{bmatrix}, \\ B_v(k) &= [v_1(k), v_2(k), c(k)]^T \end{aligned} \quad (38)$$

## 5.2 Model Predictive Control of Two AGVs

The main focus of this section is the generation of a sequence (24), which maximizes the cost function (25) taking into account the scheduling constraint (23). This section concerns the determination of the item delivery time sequence (24) for a predefined production horizon  $N_p$ . This item delivery time sequence minimizes (25) and should be determined considering both the scheduling constraints (23) and the performance of a set of  $n_v$  AGVs.

The developed framework employs a general MPC paradigm for max-plus linear systems as presented by de Schutter and van den Boom [7]. This paradigm was extended with the decision variables  $v_i(k)$ ,  $i = 1, 2$ . The main challenge is to identify the input sequence  $y(k), \dots, y(k + N_p - 1)$  on a moving horizon  $k, \dots, k + N_p - 1$ . This identification necessitates a slight modification of the cost function (25) that was previously introduced:



$$J(y) = - \sum_{j=0}^{N_p-1} y(k+j). \quad (39)$$

Consequently, the central task is obtaining  $y(k), \dots, y(k+N_p-1)$  for each  $k$ . A initial step towards the computational framework is the derivation of predictions of  $x(k+1), \dots, x(k+N_p-1)$ . This step may be realized by means of defining

$$\begin{aligned} \tilde{y}(k) &= \begin{bmatrix} y(k) \\ y(k+1) \\ \vdots \\ y(k+N_p-1) \end{bmatrix}, & \tilde{x}(k) &= \begin{bmatrix} x(k) \\ x(k+1) \\ \vdots \\ x(k+N_p-1) \end{bmatrix}, \\ \tilde{v}(k) &= \begin{bmatrix} v(k) \\ v(k+1) \\ \vdots \\ v(k+N_p-1) \end{bmatrix}, & v(k) &= [v_1(k), v_2(k)]^T. \end{aligned} \quad (40)$$

as well as a recursive application of (36). The next step, which precedes the development of the entire algorithm, is the introduction of a complete set of constraints, which is required during repetitive optimization cycles on  $k \dots, k+N_p-1$ :

**Transportation:** the transportation is defined by (31)–(34) and concerns the transportation times of a set of AGVs:

$$\begin{aligned} b_1(k) &= \max(e, b(k) + v_1(k)), \\ b_2(k) &= \max(e, b(k) + v_2(k)), \\ c_1(k) &= \max(e, c(k) + v_1(k)) \\ c_2(k) &= \max(e, c(k) + v_2(k)). \end{aligned} \quad (41)$$

**Concurrency:** concurrency is defined by (42) and pertains selecting the AGV transporting the  $k$ th item:

$$\begin{aligned} v_1(k) = e &\Leftrightarrow v_2(k) = \varepsilon \\ v_2(k) = e &\Leftrightarrow v_1(k) = \varepsilon \end{aligned} \quad (42)$$

**Scheduling:** scheduling is defined by (23) and concerns a required items delivery time:

$$x_3(k) \leq x_{ref}(k). \quad (43)$$

**Production performance:** production performance is closely connected to the maximum rate of change of the production outlet delivery time:

$$y(k+1) - y(k) \geq y_z(k), \quad (44)$$

where  $y_z(k) \geq 0$  is the production performance upper bound. Based on the preceding constraints, the complete optimization problem can be condensed to:

$$(\tilde{y}(k)^*, \tilde{v}(k)^*) = \arg \min_{\tilde{y}(k), \tilde{v}(k)} J(y), \quad (45)$$

under (41)–(44).

To conclude, the developed control strategy for two AGVs disposes of the structure given by Algorithm 6 with *fault-tolerance* capabilities.

---

**Algorithm 6:** Max-plus MPC for two AGVs

---

**Step 0:**

└ Set  $k = 1, N_p, v(0)$ ;

**Step 1:**

└ Get  $\mathbb{M}(k), \dots, \mathbb{M}(k + N_p - 1), y_z(k), \dots, y_z(k + N_p - 1)$  and  $x_{ref}(k), \dots, x_{ref}(k + N_p - 1)$  from MES;

**Step 2:**

└ Measure the state  $x(k-1)$  and obtain  $\tilde{y}(k)^*$  and  $\tilde{v}(k)^*$  by solving the constrained optimization problem (45);

**Step 3:**

└ Use the first vector elements of  $\tilde{y}(k)^*$  and  $\tilde{v}(k)^*$  (i.e.,  $y(k)^*$  and  $v(k)^*$ ) and feed them into the system (30);

**Step 4:**

└ Set  $k = k + 1$  and go to **Step 1**;

---

In addition to the rather elegant recursive description of (17) and the linearity of the cost function (39), it is possible to observe that any optimization constraint having the form  $a = \max(b, c)$  may be transformed into a set of equivalent linear constraints, i.e.,  $a \geq b, a \geq c$ . This fact obviously indicates that the optimization problem can be reduced to mixed-integer linear programming.

## 6 Fault-Tolerant Control of AGVs

The objective of this section is to provide an answer to the subsequent research question: *How to manage large inconsistencies, which may lead to the significant transportation delays and possible violation of the scheduling constraints?* This question concerns the accommodation of the possible faults, which are defined by (28). The consequence of these possible faults (28) may be a severe violation of the scheduling constraints (43). This violation can result in an infeasibility of the overall

optimization problem (17). In order to address this problem field, a time varying relaxation variable  $\alpha(k) \geq 0$  may be incorporated into (43) which results in:

$$\mathbf{x}_3(k) \leq x_{ref}(k) + \alpha(k). \quad (46)$$

In this context, it is intended that  $\alpha(k)$  is as little as possible in order to achieve a small divergence from the time schedule that was initially desired. For the purpose of obtaining the optimal values of  $\alpha_j(k)$ , a new cost function can be proposed:

$$J(\alpha) = \sum_{j=0}^{N_p-1} \alpha(k+j). \quad (47)$$

Consequently, it is possible to introduce a new FTC-oriented cost function:

$$J(y, \alpha) = (1 - \beta)J(y) + \beta J(\alpha). \quad (48)$$

In this equation,  $1 \leq \beta \leq 0$  is a constant that can be set by the control engineer and which can be adjusted to reflect the higher importance of either  $J(y)$  or  $J(\alpha)$ , respectively. By defining  $\tilde{\alpha}(k) = [\alpha(k), \dots, \alpha(k + N_p - 1)]^T$ , it is possible to rewrite the optimization problem as:

$$(\tilde{y}(k)^*, \tilde{v}(k)^*) = \arg \min_{\tilde{y}(k), \tilde{v}(k), \tilde{\alpha}(k)} J(y, \alpha), \quad (49)$$

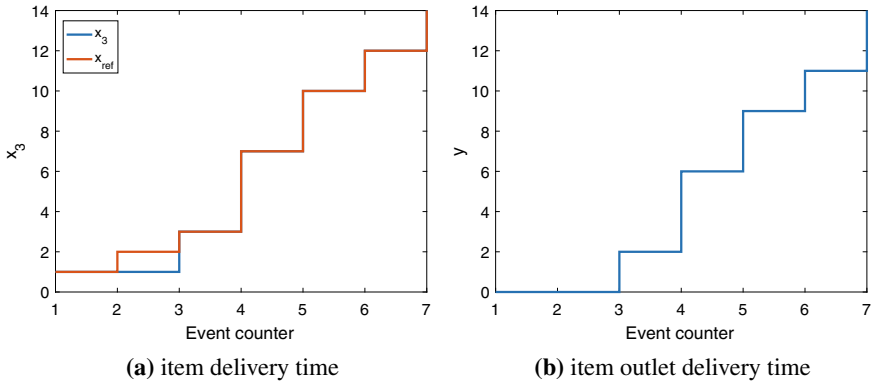
under (41)–(44) and (46). The consideration of the precedent optimization problem allows to propose an entire FTC algorithm, which updates the matrices  $A_v(\cdot, \cdot, \cdot)$  and  $B_v(\cdot, \cdot)$  together with associated constraints depending on fault estimates. FTC algorithm ensures that the optimization problem is always feasible. If the current performance of an AGV set is insufficient to attain  $x_{ref}(k)$ , it is optimally relaxed and the closest schedule to the original infeasible one is obtained.

## 6.1 Performance Evaluation

The central aim of this section is the evaluation of the reliability of *Algorithm 2* in chapter “Cyclic Two Machine Flow Shop with Disjoint Sequence-Dependent Setups”. For the sake of simplicity and clarity, it was applied to an transportation system consisting of two AGVs. The desired schedule is given by:

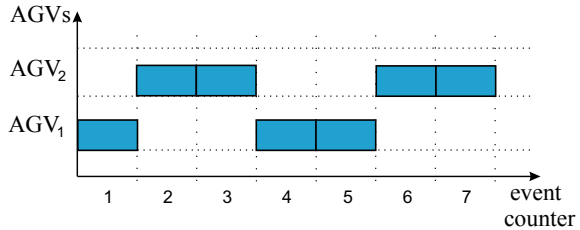
$$x_{ref} = [1, 2, 3, 7, 10, 2, 15, 16, 20]^T. \quad (50)$$

In this case, the nominal transportation times were set equal to one minute, i.e.  $b(k) = c(k) = 1$ . It is important to point out that the assumed schedule (50) is not



**Fig. 7** Comparison of  $x_3$  and  $x_{ref}$  (left) and the associated item outlet delivery time (right)

**Fig. 8** Gantt diagram of AGVs



evenly distributed, consequently it is not possible to realize the scheduling in a simple ad hoc manner. Additionally, a dedicated fault scenario was assumed, which consists of a one minute transportation delay  $f_{2,c} = 1$  of one of the AGVs during its first operation. Figure 7a shows (50) (red line) along with the actual item delivery time  $x_3$  (blue line). Figure 7b contains the respective item outlet delivery times. It is obvious that the item delivery time is larger than the desired schedule for  $k = 2$  only. Most notably, for all remaining event counters a desired schedule is achieved. Additionally, from the Gantt diagram (Fig. 8) it is evident that the second robot operates for  $k = 2$ , and therefore, according to the fault scenario a one minute delay occurs. However, the predictive FTC algorithm is able to identify this fault and can calculate a desired AGV work schedule that is able to eliminate this delay for subsequent event counters.

## 7 Remarks and Conclusions

The central research objective of this chapter was to clarify if and how a fault-tolerant interval max-plus algebra model predictive control framework can be applied for controlling flexible assembly systems which include resource conflicts. The main research contribution was the proposition of a unified FTC MPC procedure which

guarantees an optimal allocation of transportation tasks among a set of two AGVs. In particular, one of the objectives was to describe this AGV system by means of an interval max-plus algebra framework along with appropriate constraints. The proposed analytic description of two AGVs system had to be able to consider two basic properties of concurrent tasks: synchronization and concurrency. The underlying optimization criteria takes into account all transportation tasks according to a given MES-based time schedule. An decisive advantage is the fact that the cost function is linear but not quadratic. This property allows the application of the proposed approach in an on-line mode even for medium or large scale AGVs systems. The framework allows either to avoid resource conflicts or at least to minimize the possible negative influences of this kind of conflicts. The performance of this framework could be illustrated on the example of a seat assembly system, which represents all important functionalities and levels of industrial production systems. It was discussed in detail, how the design of the control and diagnosis system can enable the respective control and diagnosis task. This discussion was based on established guidelines for Design for Control. The research results allow to avoid resource conflicts in the seat assembly system and the consequences of remaining conflicts could be minimized.

**Acknowledgements** The work was supported by the National Science Centre, Poland under Grant: UMO-2017/27/B/ST7/00620.

## References

1. Rossiter, J.A.: *Model-Based Predictive Control: A Practical Approach*. CRC Press, Boca Raton (2013)
2. Prodan, I., Olaru, S., Stoica, C., Niculescu, S.-I.: Predictive control for trajectory tracking and decentralized navigation of multi-agent formations. *Int. J. Appl. Math. Comput. Sci.* **23**(1), 91–102 (2013)
3. Gruzlikov, A.M., Kolesov, N.V.: Discrete-event diagnostic model for a distributed computational system. *Merging chains. Autom. Remote Control* **78**(4), 682–688 (2017)
4. Polak, M., Majdzik, P., Banaszak, Z., Robert Wójcik, R.: The performance evaluation tool for automated prototyping of concurrent cyclic processes. *Fundam. Inf.* **60**(1), 269–289 (2004)
5. Abrams, M., Doraswamy, N., Chitra, A.M.: Visual analysis of parallel and distributed programs in the time, event, and frequency domains. *IEEE Trans. Parallel Distrib. Syst.* **3**(6), 672–685 (1992)
6. Seybold, L., Witczak, M., Majdzik, P., Stetter, R.: Towards robust predictive fault-tolerant control for a battery assembly system. *Int. J. Appl. Math. Comput. Sci.* **25**(4), 849–862 (2015)
7. De Schutter, B., Van Den Boom, T.: Model predictive control for max-plus-linear discrete event systems. *Automatica* **37**(7), 1049–1056 (2001)
8. Park, S.J., Lim, J.T.: Robust and nonblocking supervisor for discreteevent systems with model uncertainty under partial observation. *IEEE Trans. Autom. Control* **45**(9), 2393–2396 (2000)
9. Witczak, M.: *Fault Diagnosis and Fault-Tolerant Control Strategies for Non-linear Systems*. Springer International Publishing, Berlin (2014)
10. Stetter, R., Simundsson, A.: Design for control. In: *Proceedings of the 21st International Conference on Engineering Design, Vancouver, Canada, 21–25 Aug 2017*, pp. 149–158. The Design Society (2017)
11. Seybold, L., Pieczyński, A., Paczynski, A., Stetter R.: Concept of an advanced monitoring, planning, control and diagnosis system for autonomous vehicles. In: Simani, S., Bonfé, M.,

- Castaldi, P., Mimmo, N. (eds.) Proceedings of the 8th Workshop on Advanced Control and Diagnosis, ACD'2010, Ferrara, Italy, 18–19 Nov 2010, pp. 107–112 (2010)
12. Glaessgen, E., Stargel, D.: The digital twin paradigm for future NASA and US Air Force vehicles. In: 53rd AIAA/ASME/ASCE/ AHS/ASC Structures, Structural Dynamics and Materials Conference, Honolulu, Hawaii, 23–26 April 2012
  13. Tao, F.; Zhang, M.: Digital twin shop-floor: a new shop-floor paradigm towards smart manufacturing. *IEEE Access* **5**, 20418–20427 (2017)
  14. Uhlemann, H.J., Lehmann, C., Steinheipler, R.: The digital twin: realizing the cyber-physical production system for industry 4.0. *Proc. CIRP* **61**, 335–340 (2017)
  15. Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, Fangyuan, H.S.: Digital twin-driven product design, manufacturing and service with big data. *Int. J. Adv. Manuf. Technol.* **94**, 3563–3576 (2018)
  16. Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.P.: Synchronization and Linearity: An Algebra for Discrete Event Systems. Wiley, New York (1994)
  17. Butkovic, P.: Max-Linear Systems: Theory and Algorithms. Springer, Berlin (2010)
  18. Martin, H.: Transport und Lagerlogistik: Systematik, Planung, Einsatz und Wirtschaftlichkeit. Springer, Berlin (2016)
  19. Stetter, R., Bertsch, S., Eckart, P. Paczynski, A.: Torque steering system for electrical and hybrid power trains. In: Proceedings of 13th EAEC European Automotive Congress, Valencia, Spain, 13–16 June 2011 (2011)
  20. Baruwa, O.T., Piera, M.A., Guasch, A.: Deadlock-Free Scheduling Method for flexible manufacturing systems based on timed colored petri nets and anytime heuristic search *IEEE transactions on systems. Man, Cybern. Syst.* **5**(45), 1–12 (2015)
  21. An, Y., Wu, N., Chen, P.: Short-term scheduling of vehicle testing system using object petri net. *IEEE Access* **6**, 61317–61330 (2018)
  22. Ribeiro, G., Saldanha, R., Maia, C.: Analysis of decision stochastic discrete-event systems aggregating max-plus algebra and markov chain journal of control. *Autom. Electr. Syst.* **29**(5), 576–585 (2018)
  23. Majdzik, P., Akieleszak-Witczak, A., Seybold, L., Stetter, R., Mrugalska, B.: A fault-tolerant approach to the control of a battery assembly system. *Control Eng. Pract.* **55**, 139–148 (2016)

# Max-Plus Algebraic Modelling of Cyclical Multi-assortment Manufacturing System



Jarosław Stańczyk 

## 1 Introduction

In this chapter, multi-assortment production systems are considered, which can be described as *Discrete Event Systems* (DES). Due to the implementation of a certain number of products, they are characterized by repetitive, cyclical (rhythmic) behavior. Analyzing multi-assortment, cyclic production, there are a number of phenomena that have a direct impact on the behavior of systems, such as ending the production of one product or launching, in an already existing production system, the production of an additional, new product. And it is the modeling of such phenomena that is presented in this chapter.

Tools and techniques are used in DES studies, a review can be found e.g. in [3]. The theory of DES can be divided presently into three main approaches:

- *logical*—which considers the occurrence of events or the impossibility of this occurrence and the sequences of these events, but which does not consider the precise time of those occurrences i.e. does not consider performances e.g. an automata theory [5] or Petri nets [8];
- *deterministic*—which addresses the issue of performance evaluation (evaluated by the number of events occurring in a given lapse of time), and that of performance optimization e.g. the timed Petri nets or the max-plus algebra [2, 9];
- *stochastic*—used when certain statistical characteristics of the system are known, e.g. Markov chain [14], Queueing theory [18] or computer simulation.

In this work, problems of modeling a certain DES class are considered using max-plus algebra. Examples of such systems are, inter alia, production systems [10, 17]. Modeling of individual types of production systems using max-plus algebra has been presented in [6, 15].

What essentially max-plus algebra is compared to conventional algebra? The concept is based on two operators. In the max-plus algebra the addition (+) and

---

J. Stańczyk (✉)

Department of Genetics and Animal Breeding, Wrocław University of Environmental and Life Sciences, 25 C.K. Norwida St., 50-375 Wrocław, Poland  
e-mail: jaroslaw.stanczyk@upwr.edu.pl

© Springer Nature Switzerland AG 2020

W. Bożejko and G. Bocewicz (eds.), *Modelling and Performance Analysis of Cyclic Systems*, Studies in Systems, Decision and Control 241,  
[https://doi.org/10.1007/978-3-030-27652-2\\_8](https://doi.org/10.1007/978-3-030-27652-2_8)

159

multiplication ( $\times$ ) operators from the conventional algebra are replaced by the maximization ( $\max$ ) and addition ( $+$ ) operators, respectively.

The max-plus algebra was first introduced in [4]. A standard reference for long time was [1] replaced by [2, 9], a brief survey of methods and applications of this algebra is given in [7, 9]. The theory with reference to graph theory was collected in [12]. In recent years, the theory of max-plus algebra and its applications has been constantly developing, it is enough to mention a few directions in production modeling, planning and evaluation [6, 13, 15], etc. A survey of the history of max-plus algebra and its role in the field of discrete event systems is presented in [11].

All computational experiments presented in this chapter were carried out in the MATLAB environment using the *Max-Plus Algebra Toolbox for MATLAB* [16].

## 2 Problem Statement

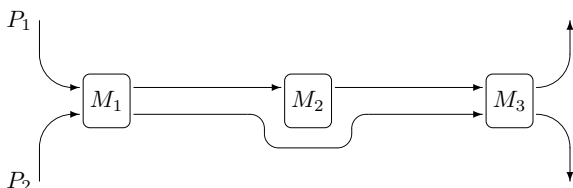
*Example 8.1* Consider a manufacturing system that consists of three machines ( $M_1$ ,  $M_2$  and  $M_3$ ). In this manufacturing system two different types of parts ( $P_1$  and  $P_2$ ) are produced according to a certain product mix. The routes followed by the various types of parts are depicted in Fig. 1.

Parts of type  $P_1$  first visit machine  $M_1$ , then  $M_2$  and then go to  $M_3$ . Parts of type  $P_2$  enter the system via machine  $M_1$ , then finally leave the system through machine  $M_3$ . It is assumed that:

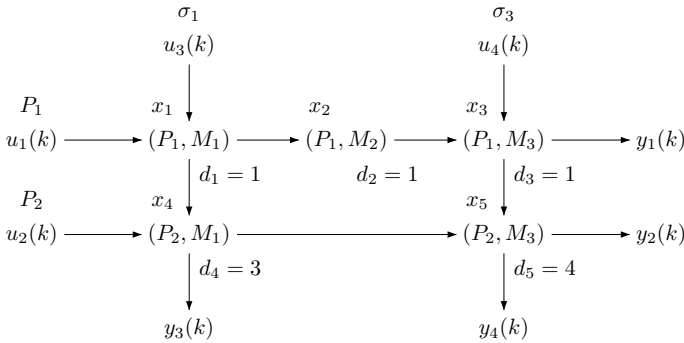
- The transportation times are negligible and that there are no set-up times on the machines when they switch from one part type to another.
- The sequencing of the various parts on the machines is known: for  $M_1$  it is  $(P_1, P_2)$ , i.e. access to machine  $M_1$  gets  $P_1$  first and then process  $P_2$ . We will call these sequences *local dispatching rules* and we will describe them as  $\sigma$  (i.e.  $\sigma_1$  for the sequence on  $M_1$ , and  $\sigma_3$  for  $M_3$ ).  $\sigma_3 = (P_1, P_2)$ . After finishing the last operation in the rule, the first one is started again, i.e. parts  $P_1, P_2$ , then  $P_1, P_2$  are processed.

We assume that the workpiece will leave the machine immediately after the operation is completed, provided that the next resources are available, and that the workpiece will be processed immediately as soon as it reaches the machine and the machine is

**Fig. 1** The routing of the various types of parts along the machines







**Fig. 2** The sequence and the duration of the various activities

free. Buffers and storehouses of sufficient capacity are located between the individual stands. The information about the sequencing and the duration of the various activities (processing times) is shown in Fig. 2. In this figure, the activities are represented by ordered pairs of the form  $(P_i, M_j)$  meaning that a part of type  $P_i$  is processed on machine  $M_j$ . The arcs represent the precedence constraints between activities. At the bottom right of each activity we have indicated its duration, e.g.  $(P_1, M_3)$  has duration  $d_3 = 1$ .

Analyzing production systems as described in Example 8.1, we consider the following problem:

1. How to model the system behavior, i.e. the implementation of individual processes over time?

This issue has been divided into:

- a. How to model a given system, depending on the availability of input materials, times of individual operations and availability of resources in order to produce workpieces of different types simultaneously?
- b. What conditions must be met for the system’s behavior to be cyclical?  
Having a model and conditions of a system whose steady state behavior is cyclical, a number of subsequent questions appear. Namely:
- c. What conditions should be met to start production in a cyclic state? It means, how to start without a transition state?
- d. What is the impact of change in the dynamics on the system behavior? I.e. how to start a new or complete an existing production process affects the length of the cycle?

### 3 Max-Plus Algebra

#### 3.1 The Basics

The max-plus algebra is defined in the field of real numbers with  $-\infty$ , i.e.:

$$\mathbb{R}_\varepsilon = \mathbb{R} \cup \{\varepsilon\} = \mathbb{R} \cup \{-\infty\} \quad (1)$$

Operators  $\oplus$  (maximum) and  $\otimes$  (plus) are defined as follows:

$$\forall a, b \in \mathbb{R}_\varepsilon \quad (a \oplus b = \max\{a, b\}), \quad (2)$$

$$\forall a, b \in \mathbb{R}_\varepsilon \quad (a \otimes b = a + b). \quad (3)$$

In addition, neutral elements are provided for individual operators:

$$\varepsilon = -\infty \quad (4)$$

and

$$e = 0 \quad (5)$$

The algebraic structure  $\mathbb{R}_{\max} = (\mathbb{R}_\varepsilon, \oplus, \otimes, \varepsilon, e)$ , is called *the max-plus algebra* or more precisely idempotent, commutative semifield.

In this chapter the notation presented in [1] is used, it means  $\varepsilon$  and  $e$  instead of  $-\infty$  and 0 respectively for emphasizing their special meanings and to avoid confusion with their roles in the conventional algebra. Additionally, notation  $ab$  instead of  $a \otimes b$  is used everywhere where it does not cause ambiguity.

Now, we extend the max-plus algebra operations to matrices in the following way. The sum  $\oplus$  of matrices  $\mathbf{A}, \mathbf{B} \in \mathbb{R}_\varepsilon^{m \times n}$  is defined to be the  $m \times n$  matrix  $\mathbf{A} \oplus \mathbf{B}$  obtained by adding corresponding entries. That is,

$$(\mathbf{A} \oplus \mathbf{B})_{ij} = (\mathbf{A})_{ij} \oplus (\mathbf{B})_{ij}, \quad i = 1, \dots, m; \quad j = 1, \dots, n. \quad (6)$$

The product  $\otimes$  of matrices  $\mathbf{A} \in \mathbb{R}_\varepsilon^{m \times p}$  and  $\mathbf{B} \in \mathbb{R}_\varepsilon^{p \times n}$  is defined to be the  $m \times n$  matrix whose  $(i, j)$ -entry is the inner product of the  $i$ th row of  $\mathbf{A}$  with the  $j$ th column in  $\mathbf{B}$ . That is,

$$(\mathbf{A} \otimes \mathbf{B})_{ij} = \bigoplus_{k=1}^p ((\mathbf{A})_{ik} \otimes (\mathbf{B})_{kj}) \equiv \max_k ((\mathbf{A})_{ik} + (\mathbf{B})_{kj}), \quad (7)$$

$$i = 1, \dots, m; \quad j = 1, \dots, n,$$

where:  $\bigoplus_{j=1}^m a_j$  is short-hand for  $a_1 \oplus \dots \oplus a_m$ .

The matrix  $\mathbf{I}_n \in \mathbb{R}_\varepsilon^{n \times n}$  with  $e$ 's on the main diagonal and  $\varepsilon$ 's elsewhere is called the *identity matrix* of order  $n$ . The matrix  $\mathbf{e} \in \mathbb{R}_\varepsilon^{m \times n}$  with  $\varepsilon_{ij} = \varepsilon$  for all  $i, j$ , is the *zero matrix*. The operator  $\star$  for square matrices  $\mathbf{A} \in \mathbb{R}_\varepsilon^{n \times n}$  is defined by:

$$\mathbf{A}^\star = \bigoplus_{k \in \mathbb{N}_0} \mathbf{A}^k, \tag{8}$$

where:  $\mathbf{A}^k = \mathbf{A} \otimes \mathbf{A}^{k-1}$ ,  $\mathbf{A}^0 = \mathbf{I}_n$ ,  $\mathbb{N}_0$  is the set of nonnegative integers.

Equation (8) is only meaningful if the right-hand side converges [4]. The operator  $\star$  (Kleene star) is used to solve the equation, where  $\mathbf{x}$  is entangled on both sides of the equation:

$$\mathbf{x} = \mathbf{A}\mathbf{x} \oplus \mathbf{b}, \tag{9}$$

so

$$\mathbf{x} = \mathbf{A}^\star \mathbf{b}. \tag{10}$$

Proof can be found e.g. in [1].

After entering the basics, let us get to the state space description.

### 3.2 State Space Description

One of the best known equation of dynamic systems is

$$\mathbf{x}(t + 1) = \mathbf{A}\mathbf{x}(t), \quad t = 1, 2, \dots, \tag{11}$$

where vector  $\mathbf{x} \in \mathbb{R}^n$  is a *state* of considered model, and matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is *state (or system) matrix*. If starting conditions are known, i.e.  $\mathbf{x}(0) = \mathbf{x}_0$ , then behavior of the system is determined. Equation (11) wrote in max-plus, where  $\mathbf{x} \in \mathbb{R}_\varepsilon^n$ ,  $\mathbf{A} \in \mathbb{R}_\varepsilon^{n \times n}$ , is as follows

$$\forall k \in \mathbb{N} \quad (\mathbf{x}(k + 1) = \mathbf{A} \otimes \mathbf{x}(k)). \tag{12}$$

In Eq. (12) instead  $t$  is  $k$ , because it is not a time of an event, but number of a cycle in which the event takes place. The most general state-space representation of a max-plus-linear system:

$$\forall k \in \mathbb{N} \quad (\mathbf{x}(k + 1) = \mathbf{A}\mathbf{x}(k) \oplus \mathbf{B}\mathbf{u}(k)), \tag{13}$$

$$(\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) \oplus \mathbf{D}\mathbf{u}(k)), \tag{14}$$

where  $\mathbf{u} \in \mathbb{R}_\varepsilon^r$ ,  $\mathbf{B} \in \mathbb{R}_\varepsilon^{n \times r}$ ,  $\mathbf{y} \in \mathbb{R}_\varepsilon^m$ ,  $\mathbf{C} \in \mathbb{R}_\varepsilon^{m \times n}$  and  $\mathbf{D} \in \mathbb{R}_\varepsilon^{m \times r}$ .

In the general case, for the  $N$ th order time-invariant system, i.e. where  $N$  previous iterations affect the current behavior of the system, with the entangled  $\mathbf{x}(k)$  on both sides of the equation, the model is represented by (15) and (16):

$$\mathbf{x}(k+1) = \bigoplus_{i=0}^{N+1} \mathbf{A}_i \mathbf{x}(k+1-i) \oplus \bigoplus_{i=0}^N \mathbf{B}_{i+1} \mathbf{u}(k-i), \quad (15)$$

$$\mathbf{y}(k) = \bigoplus_{i=0}^N \left( \mathbf{C}_{i+1} \mathbf{x}(k-i) \oplus \mathbf{D}_{i+1} \mathbf{u}(k-i) \right). \quad (16)$$

After removing  $\mathbf{x}(k+1)$  from the right side of the Eq. (15) (let assume  $\mathbf{A}_0^*$  is convergent) and after introduction of new vectors  $\tilde{\mathbf{x}}$  and  $\tilde{\mathbf{u}}$ :

$$\tilde{\mathbf{x}}(k) = [\mathbf{x}(k) \ \mathbf{x}(k-1) \ \dots \ \mathbf{x}(k-N)]^T,$$

$$\tilde{\mathbf{u}}(k) = [\mathbf{u}(k) \ \mathbf{u}(k-1) \ \dots \ \mathbf{u}(k-N)]^T,$$

and matrices:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_0^* \mathbf{A}_1 & \mathbf{A}_0^* \mathbf{A}_2 & \dots & \dots & \mathbf{A}_0^* \mathbf{A}_N \\ \mathbf{I} & \boldsymbol{\varepsilon} & \dots & \dots & \boldsymbol{\varepsilon} \\ \boldsymbol{\varepsilon} & & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \boldsymbol{\varepsilon} & \dots & \boldsymbol{\varepsilon} & \mathbf{I} & \boldsymbol{\varepsilon} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{A}_0^* \mathbf{B}_0 & \dots & \mathbf{A}_0^* \mathbf{B}_{N-1} \\ \boldsymbol{\varepsilon} & \dots & \boldsymbol{\varepsilon} \\ \vdots & & \vdots \\ \boldsymbol{\varepsilon} & \dots & \boldsymbol{\varepsilon} \end{bmatrix},$$

$$\mathbf{C} = [\mathbf{C}_0 \ \dots \ \mathbf{C}_{N-1}], \quad \mathbf{D} = [\mathbf{D}_0 \ \dots \ \mathbf{D}_{N-1}],$$

where  $\mathbf{I}$  i  $\boldsymbol{\varepsilon}$  are appropriate size max-plus-algebraic identity and zero matrix, 1st order model, described by (13) and (14), is obtained.

## 4 Multi-assortment Manufacturing System

*Example 8.2* Let's continue with the considerations of the Example of 8.1. What does the max-plus model of an algebraic model look like? How the execution of individual processes is represented based on such model?

In order to simplify the process of deriving the evolution equations of this system, we shall first look at what happens in one cycle of the production process. We define:

- $u_i(k)$  time instant at which the raw material for a part of type  $P_i$  is available in the  $k$ th production cycle for  $i = 1, 2$ ;
- $u_j(k)$  time instant at which machine  $M_1$  and  $M_3$  is available for the first activity that should be performed on it in the  $k$ th production cycle for  $j = 3, 4$ ;
- $x_i(k)$  time instant at which activity  $i$  starts in the  $k$ th production cycle for  $i = 1, 2, \dots, 5$ ;
- $y_i(k)$  time instant at which the finished product of type  $P_i$  of the  $k$ th production cycle has been completed for  $i = 1, 2$ ;

$y_j(k)$  time instant at which machine  $M_3$  (or  $M_4$ ) has finished processing the last part of the  $k$ th production cycle that should be processed on it for  $j = 3, 4$ .

We have the following evolution equations:

$$\begin{aligned} x_1(k+1) &= d_1x_1(k) \oplus u_1(k) \oplus u_3(k), \\ x_2(k+1) &= d_1x_1(k+1) \oplus d_2x_2(k), \\ &\vdots \end{aligned} \tag{17}$$

or, more compactly in matrix description:

$$\mathbf{x}(k+1) = \mathbf{A}_0\mathbf{x}(k+1) \oplus \mathbf{A}_1\mathbf{x}(k) \oplus \mathbf{B}_1\mathbf{u}(k) \tag{18}$$

$$= \mathbf{A}\mathbf{x}(k) \oplus \mathbf{B}\mathbf{u}(k), \tag{19}$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k), \tag{20}$$

where  $\mathbf{A} = \mathbf{A}_0^*\mathbf{A}_1$  and  $\mathbf{B} = \mathbf{A}_0^*\mathbf{B}_1$  and

$$\mathbf{x}(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \\ x_5(k) \end{bmatrix}, \quad \mathbf{A}_0 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ d_1 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & d_2 & \varepsilon & \varepsilon & \varepsilon \\ d_1 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & d_3 & d_4 & \varepsilon \end{bmatrix}, \quad \mathbf{A}_1 = \begin{bmatrix} d_1 & \varepsilon & \varepsilon & d_4 & \varepsilon \\ \varepsilon & d_2 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & d_3 & \varepsilon & d_5 \\ \varepsilon & \varepsilon & \varepsilon & d_4 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & d_5 \end{bmatrix},$$

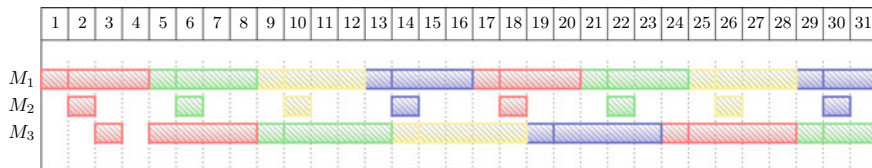
$$\mathbf{B}_1 = \begin{bmatrix} 0 & \varepsilon & 0 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 0 \\ \varepsilon & 0 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \varepsilon & \varepsilon & d_3 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & d_5 \\ \varepsilon & \varepsilon & \varepsilon & d_4 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & d_5 \end{bmatrix}, \quad \mathbf{u}(k) = \begin{bmatrix} u_1(k) \\ u_2(k) \\ u_3(k) \\ u_4(k) \end{bmatrix}, \quad \mathbf{y}(k) = \begin{bmatrix} y_1(k) \\ y_2(k) \\ y_3(k) \\ y_4(k) \end{bmatrix}.$$

After substituting numerical values we get:

$$\mathbf{A} = \begin{bmatrix} 1 & \varepsilon & \varepsilon & 3 & \varepsilon \\ 2 & 1 & \varepsilon & 4 & \varepsilon \\ 3 & 2 & 1 & 5 & 4 \\ 2 & \varepsilon & \varepsilon & 4 & \varepsilon \\ 5 & 3 & 2 & 7 & 5 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & \varepsilon & 0 & \varepsilon \\ 1 & \varepsilon & 1 & \varepsilon \\ 2 & \varepsilon & 2 & 0 \\ 1 & 0 & 1 & \varepsilon \\ 4 & 3 & 4 & 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} \varepsilon & \varepsilon & 1 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 3 \\ \varepsilon & \varepsilon & \varepsilon & 3 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 3 \end{bmatrix}.$$

Performing processes in such a system is shown in the Gantt chart in Fig. 3. For the purposes of simulation, it was assumed that at the moment of take-off all machines and inter-operation buffers are empty, i.e.

$$\mathbf{x}(0) = \boldsymbol{\varepsilon}. \tag{21}$$



**Fig. 3** The Gantt's chart of machine occupancy in manufacturing system from Example 8.2 for production of the first few parts

Hence

$$\mathbf{x}(1) = \mathbf{B}\mathbf{u}(0). \tag{22}$$

Additionally, the input material is always available, i.e.

$$\mathbf{u}(k) = 0. \tag{23}$$

The Fig. 3 shows the first few iterations. First one, marked in red. At the moment of take-off the operation  $P_1$  on  $M_1$  is performed. Then the workpiece goes to  $M_2$ , and working of workpiece  $P_2$  starts on  $M_1$ . After completing the operation on  $M_2$ , the workpiece goes to the machine  $M_3$  (idle up to now) and leaves the system after the operation is completed. The  $M_3$  remains idle for one time instant until it starts processing  $P_2$ . After leaving  $P_2$  of the machine  $M_1$ , the second iteration starts—marked in green. The second iteration starts with the processing of workpiece  $P_1$  on  $M_1$ , and so on.

Modeling of stationary systems, described by Eqs. (19)–(20), was among others the topic of the work [17]. There, the problem of modeling inter-operation buffers and the impact of buffer capacity on system behavior were also considered.

### 4.1 Cyclic Systems

In cyclic systems, there is a *certain* relationship between the system outputs and its inputs, which creates a closed system, with production taking place rhythmically. So, it is advisable to introduce a matrix  $\mathbf{K} \in \mathbb{R}^{r \times m}$ , to describe the dynamics of restarting the system for the next cycle (and cyclical dependence of inputs and outputs):

$$\mathbf{u}(k + 1) = \mathbf{K}\mathbf{y}(k). \tag{24}$$

Hence:

$$\mathbf{x}(k + 1) = \mathbf{A}\mathbf{x}(k) \oplus \mathbf{B}\mathbf{K}\mathbf{y}(k) \tag{25}$$

$$= \mathbf{A}\mathbf{x}(k) \oplus \mathbf{B}\mathbf{K}\mathbf{C}\mathbf{x}(k) \tag{26}$$

$$= (\mathbf{A} \oplus \mathbf{BKC})\mathbf{x}(k) \tag{27}$$

$$= \hat{\mathbf{A}}\mathbf{x}(k). \tag{28}$$

So, we obtain an autonomous model.

*Example 8.3* Let's go back to the example from the beginning of this section and introduce an additional assumption: there is only one pallet available for each type of part. Which means that only after the part is finished, when the pallet leaves the system, the next pallet will be able to enter the system. Hence:

$$\hat{\mathbf{A}} = \mathbf{A} \oplus \mathbf{BKC} \tag{29}$$

$$= \begin{bmatrix} 1 & \varepsilon & \varepsilon & 3 & \varepsilon \\ 2 & 1 & \varepsilon & 4 & \varepsilon \\ 3 & 2 & 1 & 5 & 3 \\ 2 & \varepsilon & \varepsilon & 4 & \varepsilon \\ 5 & 3 & 2 & 7 & 4 \end{bmatrix} \oplus \begin{bmatrix} 0 & \varepsilon & 0 & \varepsilon \\ 1 & \varepsilon & 1 & \varepsilon \\ 2 & \varepsilon & 2 & 0 \\ 1 & 0 & 1 & \varepsilon \\ 4 & 3 & 4 & 1 \end{bmatrix} \begin{bmatrix} 0 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 0 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 0 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 0 \end{bmatrix} \begin{bmatrix} \varepsilon & \varepsilon & d_3 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & d_5 \\ \varepsilon & \varepsilon & \varepsilon & d_4 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & d_5 \end{bmatrix} \tag{30}$$

$$= \begin{bmatrix} 1 & \varepsilon & 1 & 3 & \varepsilon \\ 2 & 1 & 2 & 4 & \varepsilon \\ 3 & 2 & 3 & 5 & 3 \\ 2 & \varepsilon & 2 & 4 & 3 \\ 5 & 3 & 5 & 7 & 6 \end{bmatrix} \tag{31}$$

Individual operations in the system are shown in Fig.4. There is assumed, as in the previous simulation, that  $\mathbf{x}(0) = \varepsilon$ . As can be seen, that execution of processes differs from Fig. 3.

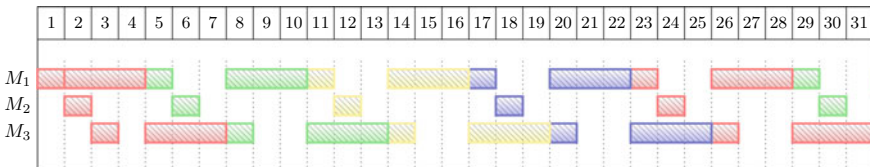
On the other hand, a discrete system can be treated as cyclic, if the following condition holds:

$$\exists k_0 \in \mathbb{N}_0, T \in \mathbb{R}_\varepsilon \forall k \geq k_0 \quad (\mathbf{x}(k + 1) = T\mathbf{x}(k)), \tag{32}$$

where  $T$  is a cycle time (period),  $k_0$  is length of transient state.

The most useful max-plus practicable results are relative to the spectral problem (33).

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v} \tag{33}$$



**Fig. 4** The Gantt's chart of machine occupancy in manufacturing system from Example 8.3

where:  $\lambda$  is eigenvalue of  $\mathbf{A}$ ,  $\mathbf{v}$  is eigenvector associated with  $\mathbf{A}$ . The theory is very similar to the theory of Perron–Frobenius when it comes to eigenvalue and eigenvector.

**Theorem 8.1** (The Perron–Frobenius Theorem) *An irreducible matrix  $\mathbf{A} \in \mathbb{R}_\varepsilon$  has unique eigenvalue, equal to the maximal circuit mean of  $\mathbf{A}$ .*

Unlike in conventional P-F. theory an irreducible max-plus algebraic matrix may have several (non proportional) eigenvectors, hence there are a number of algorithms for determining these vectors.

**Definition 8.1** (Cyclicity of a matrix) *An irreducible matrix  $\mathbf{A} \in \mathbb{R}_\varepsilon^{n \times n}$  is cyclic if:*

$$\exists d, m, M \in \mathbb{N}_0 \forall m \geq M \quad (\mathbf{A}^{d+m} = \lambda^d \mathbf{A}^m), \quad (34)$$

where  $\lambda$  is maximum cycle mean of  $\mathbf{A}$ , i.e.

$$\lambda = \bigoplus_{i=1}^n (\text{trace}(\mathbf{A}^i)^{\frac{1}{i}}), \quad (35)$$

$$\text{trace}(\mathbf{A}) = \bigoplus_{j=1}^n a_{jj}, \quad (36)$$

and  $d$  is quantity of arcs in critical circuit.

If system matrix  $\mathbf{A}$  is cyclic, then the system is cyclic, and the length of period  $T$  in steady state:

$$T = \lambda^d \quad (37)$$

For the Example 8.3, where the implementation of the processes has been presented in Fig. 4, and matrix  $\mathbf{A}$  has been described by the Eq. (31):  $T = 6$  (i.e.  $\lambda = 6$  and  $d = 1$ ).

In the general case, for the system described by Eq. 13, behavior of the system may be cyclic or not, depending on the second part of the equation, more specifically depends on  $\mathbf{u}(k)$ . If the input vector changes cyclically, as presented by the Eq. 38

$$\mathbf{u}(k + 1) = T_u \mathbf{u}(k), \quad (38)$$

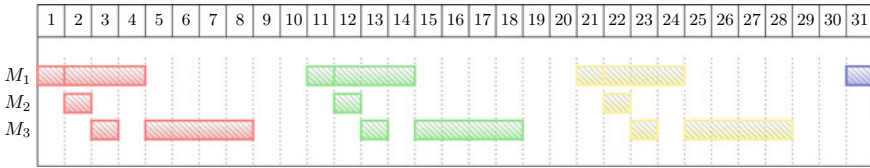
and the condition (39) is met, then the system is cyclic.

$$T \leq T_u, \quad (39)$$

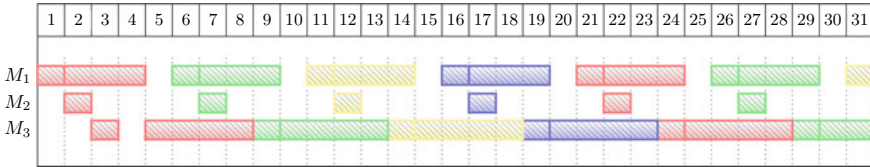
where  $T$  is a period of cyclicity of system matrix  $\mathbf{A}$ .

For  $T > T_u$ , although the input vector behaves cyclically, however, the system will not keep up with the processing of the input material, therefore, the system will not behave cyclically. Example 8.4 shows different system behaviors depending on  $T_n$ .

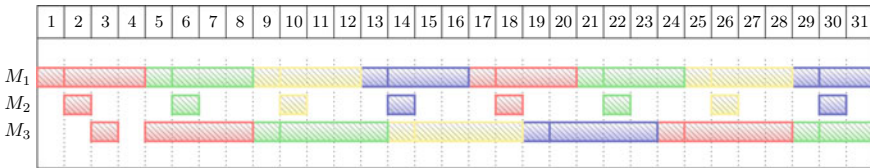




**Fig. 5** The Gantt’s chart of processes execution with  $T_u = 10$



**Fig. 6** The Gantt’s chart of processes execution with  $T_u = 5$



**Fig. 7** The Gantt’s chart of processes execution with  $T_u = 4$

*Example 8.4* Below have been presented 3 different implementations of processes in the production system of Example 8.2. They differ only in times, when the input material goes to the system (i.e. inputs vectors are different). In this example  $T = 5$ .

- (a) Figure 5 shows the execution of processes when the input material appears every 10 time units. System is cyclic, with  $T_u = 10$ .
- (b) Figure 6 presents the Gantt’s chart, when the input material appears every 5 time units. System is cyclic,  $T_u = T = 5$ .
- (c) Figure 7 presents the system behavior, when the input material appears every 4 time units. System is not cyclic ( $T_u < T$ ), in each iteration the occupation of the staging buffers increases, and operations on the machine  $M_3$  begins later. The same behavior can be observed in Fig. 3, where  $\mathbf{u}(k) = 0$ .

### 4.2 System Without Transient State

What conditions must be fulfil, in order to processes execution begin in steady state?

To start with, consider the autonomous system, i.e. as described by the Eq. (12) or (28). In other words:

$$\mathbf{x}(k + 1) = \mathbf{A}\mathbf{x}(k) \tag{40}$$

$$= T\mathbf{x}(k) \tag{41}$$

$$= T^{k+1}\mathbf{x}(0). \tag{42}$$

Hence:

1. the matrix  $\mathbf{A}$  must be cyclic (with period  $T$ ),
2. the initial vector  $\mathbf{x}(0)$  must be an eigenvector of the matrix  $\mathbf{A}$ .

Consider the case of a system in which his behavior determines the matrix  $\mathbf{A}$ , but the initial state depends on the inputs—as in the Eqs. (21) and (22). Then, besides the cyclicity of the matrix  $\mathbf{A}$ , the condition must be fulfilled:

$$\mathbf{B}\mathbf{u}(0) = \mathbf{v}, \tag{43}$$

where  $\mathbf{v}$  is an eigenvalue of the matrix  $\mathbf{A}$ .

The designation of such  $\mathbf{u}(0)$  to meet the Eq. (43) is not possible in the general case. Which means that the system before it reaches the determined cyclical state, in which the execution of the processes is rhythmic, will have so-called transient (starting) state.

### 4.3 Ending Production

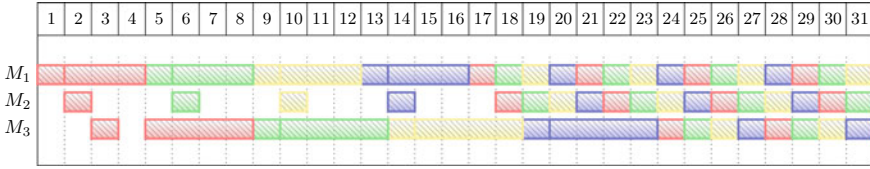
In this section, we consider a situation in which, after producing of the desired amount of elements of a given type, we want to finish producing that elements. How will this affect the behavior of the modeled system?

Ending of the production of any of the elements is associated with a change in the parameters of the model. We assume that extinction the production of a selected element means stopping the production of new elements, while production in progress is continued. By changing the production dynamics, by adding or removing the production process, the system model dynamically changes Let’s look at the example below.

*Example 8.5* Let’s continue with the considerations of the production system from Example 8.2. As in Example 8.2, we start productions under the same initial conditions, i.e.  $\mathbf{x}(0) = [\varepsilon \ \varepsilon \ \varepsilon \ \varepsilon \ \varepsilon]^T$ , and  $\mathbf{u}(k) = [0 \ 0 \ 0 \ 0]^T$ . Suppose, that after producing the appropriate number of products e.g. 4 items, of the type  $P_2$ , this production process is ending. How will the production schedule look like then? For  $k \leq 4$ , the system model is exactly the same as in Example 8.2. The model changes for  $k \geq 5$ :

System matrix  $\mathbf{A} = \mathbf{A}_0^* \mathbf{A}_1$ , where

$$\mathbf{A}_0 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ d_1 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & d_2 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix}, \quad \mathbf{A}_1 = \begin{bmatrix} d_1 & \varepsilon & \varepsilon & d_4 & \varepsilon \\ \varepsilon & d_2 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & d_3 & \varepsilon & d_5 \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix}.$$



**Fig. 8** The Gantt's chart from Example 8.5

Input matrix  $\mathbf{B} = \mathbf{A}_0^* \mathbf{B}_1$  and input vector

$$\mathbf{B}_1 = \begin{bmatrix} 0 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix}, \quad \mathbf{u}(k) = \begin{bmatrix} 0 \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{bmatrix}.$$

Output matrix and output vector:

$$\mathbf{C} = \begin{bmatrix} \varepsilon & \varepsilon & 1 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix}, \quad \mathbf{u}(k) = \begin{bmatrix} y_1 \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{bmatrix}.$$

The processes execution in the system is presented as the Gantt's chart on Fig. 8.

In conclusion, adding changes to the model described by Eqs. (13)–(14) induce by production changes, we obtain a model in which not only individual vectors depend on production iterations but also matrices, that is, we obtain a time-variant model:

$$\mathbf{x}(k + 1) = \mathbf{A}(k)\mathbf{x}(k) \oplus \mathbf{B}(k)\mathbf{u}(k), \tag{44}$$

$$\mathbf{y}(k) = \mathbf{C}(k)\mathbf{x}(k) \oplus \mathbf{D}(k)\mathbf{u}(k). \tag{45}$$

## 5 Remarks and Conclusions

Max-plus algebra is a convenient analytical tool for modeling the behavior of production systems. Thanks to its use, it is easy to obtain a number of quantitative data regarding, e.g. resource utilization over time. Directly from matrix  $\mathbf{A}$ , a set of initial state vectors can be obtained for which production processes will be started immediately in a steady state (without transition state). In addition, for stationary systems, analysis of matrix  $\mathbf{A}$  provides a number of information, such as the length of the cycle in steady state. The non-stationary model, obtained in the case of introducing dynamics related to production changes, complicates this situation, but extends the class of systems that can be effectively modeled using max-plus algebra.

## References

1. Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.P.: Synchronisation Linearity. Wiley, New York (1992)
2. Butkovič, P.: Max-linear Systems: Theory and Algorithms. Springer, Berlin (2010)
3. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems, 2nd edn. Springer, Berlin (2007)
4. Cuninghame-Green, R.: Minimax Algebra. Lecture Notes in Economics and Mathematical Systems, vol. 166. Springer, Berlin (1979)
5. Darabi, H., Jafari, M., Manapure, S.: Finite automata decomposition for flexible manufacturing systems control and scheduling. *IEEE Trans. Syst. Man Cybern.* **33**(2), 168–175 (2003)
6. David-Henriet, X., Hardouin, L., Raisch, J.: Max-Plus-Linear Systems for Modeling and Control of Manufacturing Problems. In: Ghezzi, L., Hömberg, D., Landry, C. (eds.) *Math for the Digital Factory*, pp. 37–60. Springer (2017)
7. Gaubert, S.: Max-Plus: Methods and Applications of (max, +) Linear Algebra. Technical Report RR-3088, INRIA. <https://hal.inria.fr/inria-00073603> (1997). Accessed 29 March 2019
8. Girault, C., Valk, R.: Petri Nets for Systems Engineering. Springer, Berlin (2003)
9. Heidergott, B., Olsder, G.J., van der Woude, J.: Max Plus at Work: Modeling and Analysis of Synchronized Systems. Princeton University Press, Princeton (2005)
10. Indriyani, D., Subiono, S.: Scheduling of the crystal sugar production system in sugar factory using max-plus algebra. *Int. J. Comput. Appl. Math.* **2**(3), 33–37 (2016)
11. Komenda, J., Lahaye, S., Boimond, J.L., van den Boom, T.J.: Max-plus algebra in the history of discrete event systems. *Ann. Rev. Control* **45**, 240–249 (2018)
12. León, F.P., Kiencke, U.: Ereignisdiskrete Systeme. Oldenbourg Verlag (2013)
13. Nambiar, A.N., Imaev, A., Judd, R.P., Carlo, H.J.: Production Planning Models using Max-Plus Algebra. In: Modrák, V., Pandian, R.S. (eds.) *Operations Management Research and Cellular Manufacturing Systems*, pp. 227–257. IGI Global (2012)
14. Papadopoulos, C.T., Li, J., O’Kelly, M.E.: A classification and review of timed markov models of manufacturing systems. *Comput. Ind. Eng.* **128**(1), 219–244 (2019)
15. Seleim, A., El Maraghy, H.: Max-plus modeling of manufacturing flow lines. In: *Proceedings of 47th CIRP Conference on Manufacturing Systems (CMS2014)*, vol. 17, pp. 302–307 (2014). <https://doi.org/10.1016/j.procir.2014.01.133>
16. Stańczyk, J.: Max-Plus Algebra Toolbox for Matlab, ver. 1.7. <http://gen.up.wroc.pl/stanczyk/mpa/> (2016). Accessed 29 March 2019
17. Stańczyk, J.: Max-plus algebra as a tool for the modelling and performance analysis of manufacturing systems. *Oper. Res. Decis.* **28**(3), 77–97 (2018). <https://doi.org/10.5277/ord180307>
18. Yu, A.J.: Queueing Theory Applications in Manufacturing Systems. In: Bhat, U.N. (ed.) *An Introduction to Queueing Theory*, pp. 239–253. Springer (2015)

# **Petri Nets for Cyclic Systems Modeling**

# Incorporating Automatic Model Checking into GPenSIM



Reggie Davidrajuh , Bozena Skolud  and Damian Krenczyk 

**Abstract** Large-scale manufacturing systems involve hardware and software that are highly interconnected and complex. Unexpected failures in these systems can cause material damages and can risk human lives too. The definite way of avoiding unexpected failures is to make a model of the system and to perform model verification and validation on it. Petri nets are a highly effective way of modelling discrete-event systems. Model checking is the terminology that is used for model verification on Petri Nets. General-purpose Petri Net Simulator (GPenSIM) is a tool for modelling, simulation, performance evaluation, and control of discrete-event systems (GPenSIM: a general purpose Petri net simulator, <http://www.davidrajuh.net/gpensim>, 2019, [15]). GPenSIM is developed by one of the authors of this chapter. This chapter explores the potentials of incorporating the model checking functions to GPenSIM. In this chapter, the problem of model checking is presented. The chapter introduces Activity-Oriented Petri Nets (AOPN) and GPenSIM for model checking of cyclic production systems.

## 1 Cyclic Concurrent Processes

High competitiveness between manufacturers is a great challenge in this age. Undoubtedly it always boils down to increasing the offer. This is the reason for looking for solutions, which, like the group technology will enable the production

---

R. Davidrajuh

Department of Electrical & Computer Engineering, University of Stavanger,  
4036 Stavanger, Norway

e-mail: [reggie.davidrajuh@uis.no](mailto:reggie.davidrajuh@uis.no)

B. Skolud (✉) · D. Krenczyk

Faculty of Mechanical Engineering, Institute of Engineering Processes Automation  
and Integrated Manufacturing Systems, Silesian University of Technology,  
18A Konarskiego Str., 44-100 Gliwice, Poland

e-mail: [bozena.skolud@polsl.pl](mailto:bozena.skolud@polsl.pl)

D. Krenczyk

e-mail: [damian.krenczyk@polsl.pl](mailto:damian.krenczyk@polsl.pl)

© Springer Nature Switzerland AG 2020

W. Bożejko and G. Bocewicz (eds.), *Modelling and Performance Analysis of Cyclic Systems*, Studies in Systems, Decision and Control 241,  
[https://doi.org/10.1007/978-3-030-27652-2\\_9](https://doi.org/10.1007/978-3-030-27652-2_9)

of small series and even individual products in repetitive manufacturing conditions, which in turn allows focusing on the repetition period, instead of the whole production. In this context, cyclic production is becoming more and more popular. Cyclic scheduling reduces inventory levels. It also allows manufacturing using common resources that are technologically diverse, at the same time ensuring a high level of utilization. Recently, the tendency to produce of many different products in one production system is observed. However from the operational point of view all those operations that should be executed on one machine are similar one to the another one, so that there is no need to preset machine between those operations. This situation is often observed e.g. in automotive industry, household goods production, etc. Cyclic multi-assortment production (cyclic job shop) is a topic present in many works.

Rhythmic production is a form of production organization that enables analytical determination of the indicators that characterize it. With regard to the production process, rhythmicity is understood as regular, repetition of specific elements of the production process (operations, activities). The rhythmic production system is the one in which the completion of the last operation in the sequence results in the return to the first in the sequence. The rhythmic production system is usually used to produce one assortment. In recent years, the demand for this type of production has been declining. However, it is important that the rhythmic organization of production allows the determination of the parameters that characterize the production at a near-optimal level. Therefore, rhythmicity is also used in the production of many assortments. Problems related to conducting rhythmic production are a predicate of many scientific studies. The authors of this chapter also dealt with these issues. Analytical models are used for descriptions of rhythmic systems, being formal models with a mathematical basis, such as queuing theory, mathematical programming, computer simulation, Petri Nets, max+ algebra, minimax algebra, etc.

In this chapter a cyclic behavior in steady state is considered. The model is deterministic. The deterministic model is a mathematical model which precisely determines known relationships among states and events, without any room for random variation. In such models, a given input will always produce the same output.

The advantage of the cyclical approach to production is that it has a lower degree of complexity compared to the general problem complexity. The created model for a single production cycle is easier to analyze. Modeling of the complex production system is easy to miss important interaction patterns. Taking into account the complexity of concurrent production systems it is important to provide methods for its checking, methods that enable its testing before building and running the system. The problem is how to design the system to be sure of its functioning. The need is to build an executable model of the system [17].

Korbaa et al. [20] presented the problem of determination of command of FMS for small and medium production, and chosen cyclic behavior to reduce complexity. The aim was to reduce WIP and reach optimal production speed. They choose few heuristics (Hillion's [16], Valentin's [26], and Ohl's [24] Heuristic) and conclude that the complexity is large and computation time is large too. Authors propose to limit some parameters to shorten computational time (depth of the search tree or number of intervals for each machine) but are not fit for real production conditions.

In the paper [1] the evaluation of transfer function in this capability of identifying and describing the dynamics of a project-driven system in repetitive production systems in a repetitive process in construction. Authors proposed black-box modeling, trial and error method where parameters of various models are estimated, and the output of these models are compared to the results with the opportunity for further refinement. Hillion and Proth [16] used event graphs to model cyclic systems with their control. They showed that it is theoretically possible to reach the optimal cycle time (given by the bottleneck machine) and proposed an algorithm to meet the performance while minimizing the WIP. Valentin [26] revisited the previous approach and introduced available intervals to improve the previous results [21].

### Reachability

A reachability problem consists of checking whether a given set of target states can be reached starting from a fixed set of initial states. The set of target states can be represented explicitly or via some implicit representation (e.g., a system of equations, a set of minimal elements with respect to some ordering on the states). Authors of [18] proposed Petri net as a tool which is a simple yet powerful formalism for representation of concurrency and interaction of events in a system. The reachability problem with Petri nets implementation is discussed in this paper. Authors conclude in this work that many problems concerning Petri net behavior are intrinsically very hard to solve. Unfortunately in the discussed problem they require unacceptable amounts of computation time or space. In [2] some decision problems related to the reachability problem for Petri nets are presented. Authors prove that for Petri nets reachability problem is equivalent to equality problem. In some Petri nets the reachability problem is undesirable for generalization of Petri nets in which some transitions can reset a certain place to zero marking. Authors of [3–5] presented the reachability problem in a system of repetitive concurrent processes which is treated as a system of autonomous processes that compete for access to shared resources. The access is controlled by dispatching rules allocated to the resources, which handle the required synchronization of the process execution. They discussed a problem of initial processes and dispatching rule allocation enables to predict the deadlock-free and starvation-free behavior. In [3] the policy of the buffer capacity allocation is discussed for the sufficient condition guaranteeing a cyclic steady state behavior for some combination of initial state and priority rules allocation including resource capacity allocation. In [4, 6], the multimodal processes are defined. Operations are executed along sequences that repeat an indefinite number of times. Train or bus traffic can be considered as an example of such kind of systems. A concept of multimodal processes system is presented by authors of [4, 5]. Each process route can interact with other processes by so-called common resources. In general each process is executed repetitively State space  $S^*$  composed of the sequence of resources allocation  $A_k$  state  $S_t$  can be linked via another state by transitions. Authors show the cyclic scheduling problem of multimodal processes characteristic for the transportation network. Authors propose local dispatching rule, and they create a schedule showing time-dependent activities (multimodal process behavior) which describe detailed dependence of each one activity with others. The problem is NP-hard. The approach



shows that the most important advantage (distributed control) is lost. In addition, this work presents an approach which is based on recognizing all possible states and checking the known state from the whole states; the checking is time-consuming and in more complicated problems is practically useless. Wójcik [27] states that the initial state and set of dispatching rules can be seen as control variables allowing to adjust processes schedule. “Switching” among cyclic steady state can be modeled in terms of constraints satisfaction problem and implemented in declarative language environment OZ Mozart system. The same author in [28] described a system of repetitive manufacturing processes as a set of process sharing common resources using a mutual exclusion protocol. The problem of designing no-wait cyclic schedule for this class of repetitive manufacturing can be solved using constraints programming. Constraints propagation is an efficient inference mechanism that is intended to narrow the variable domains. The idea is based on a logical analysis of the constraints to derive the new constraints, which define a smaller space of the admissible solution. The method of constraint propagation reduces the size of search space. The problem was solved by checking the starting time for the processes for which the no-wait cyclic schedule exist. The paper [9] deals with the problem of state space in cyclic systems. The state space explosion often makes verification impractical. According to [9] two method can be can be used for such. One is the state space reduction. Second is use algorithms which traverse the state space in a more efficient manner. Cheng et al. [9] proposed to improve a standard linear time model checking algorithm by taking into account strongly connected components.

Reachability is a fundamental basis for studying the dynamic properties of any system. Building a Petri network based on an informal or even formal program specification is a difficult issue. In many cases, the process of building a model in the form of a Petri net reveals the incompleteness of the specification. This is important in systems for critical applications. The method is based on the construction of the reachability tree. From the  $M_0$  state all possible transitions that lead to reachable markings forming nodes of the graph are fired, with successive ones, etc. It has been shown that the reachability problem take at least exponential space to verify in a general case [22, 25].

A different modification of Petri net is proposed to tackle the exponential space problem. However, the equality problem is undecidable, i.e., there is no algorithm for determining if firing sequences for any two Petri nets are equal. That’s why authors propose another tool GPenSIM which based on Activity-oriented Petri Nets and elaborated by the first author of the chapter.

## 2 Activity-Oriented Petri Net Models of Cyclic Systems

It seems Petri nets are very useful for modeling the repetitive production planning problem as the problem is basically a discrete system. However, due to a large number of production resources involved in the repetitive production planning problem, the resulting Petri net model will become huge. Usually, even for a simple problem with a

few resources, the resulting Petri net model is very large [10]. Activity-oriented Petri Nets (AOPN) is a methodology that provides smaller Petri net models of systems that involves a large number of resources [10, 12].

The model building with AOPN is performed in two steps [13]: In the first step (phase-I), the static Petri Net graph is developed. In the first step, only the activities considered for developing the Petri Net graph, and the resources are not considered. By taking only the activities (which will be represented by transitions in the Petri Net graph), a smaller Static Petri Net graph is yielded.

In the second step (phase-II), the run-time dynamic model is developed. In this step, the following run-time details are considered: transitions (activities) requesting the resources, consuming the resources if they allocated, and then releasing the resources after completion of the activity.

### The Run-Time Petri Net Model Example

The considered, repetitive production planning problem involves three processes, having three, four, and two activities, respectively. Also, there are four resources involved in the problem. By the phase-I Developing the static Petri net graph, the static Petri net graph shown in the Fig. 1 is obtained, which only possess the activities of the three processes and the precedence relationship between them. In the phase-II Developing the run-time model, all the run-time details are added to the static Petri net graph to make it as a dynamic model. For example, the following dynamic details are added to the model:

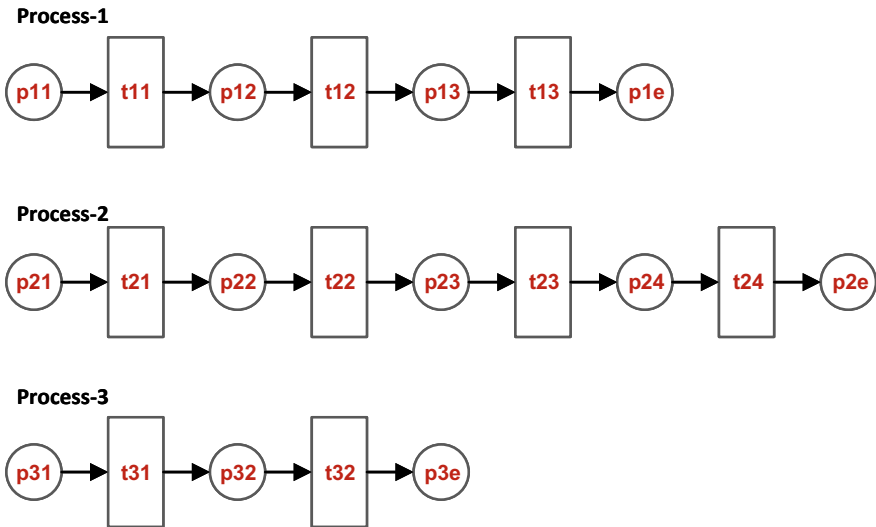


Fig. 1 Static Petri net graph of the example

- The connection between the activities and the resources: The Resource-Allocation-Policy (RAP) defines the set of resources that can perform the specific activities of a process ( $mp1,1$  to  $mp1,n$ ).
- The expected duration of the activities (cycle times): The cycle time ( $mp2, j$ ) of the process- $i$  becomes the firing times of the respective transition  $tij$ .

### 3 General-Purpose Petri Net Simulator

General-purpose Petri net simulator (GPenSIM) is a MATLAB toolbox. GPenSIM is for modeling, simulation, performance analysis, and control of discrete systems. GPenSIM is relatively new software. However, GPenSIM is already being used by some universities [7, 8, 19, 23]. The users report that the reasons for using GPenSIM is its simplicity of learning and using, and its flexibility to create newer functionality [7, 8, 11, 14, 19, 23]. Implementing a Petri net model with GPenSIM usually results in four M-files [11, 14]:

1. Petri net Definition File (PDF): A PDF declares the static Petri net graph: the set of places, the set of transitions, and the set of arcs are declared in this file.
2. Main Simulation File (MSF): The MSF declares the initial dynamics (e.g., initial tokens in the places, firing times of the transitions, firing costs of the transitions) and runs the simulations. When the simulation terminates, the code for plotting and printing the simulation results are also coded in this file.
3. The pre-processor file (COMMON\_PRE): If there are additional conditions for the enabled transitions to satisfy before firing, these conditions are coded in the COMMON\_PRE file.
4. The post-processor file (COMMON\_POST): If there are any post-firing actions to be performed after firing of transitions, these actions can be coded in the COMMON\_POST file.

For model checking (discussed in the following section), the reachability graph has to be generated. This is done by calling the function ‘cotree’ in the MSF.

### 4 Automatic Model Checking Method

In this section, a running example is used to show how automatic model checking can be done for repetitive production systems.

#### Running Example:

The running example is shown in Fig. 2, which involves two CNC machines and a robot. The operational specifications of the system:

1. To start a cycle, a raw part must be available on the incoming conveyor belt, and the robot is also available.

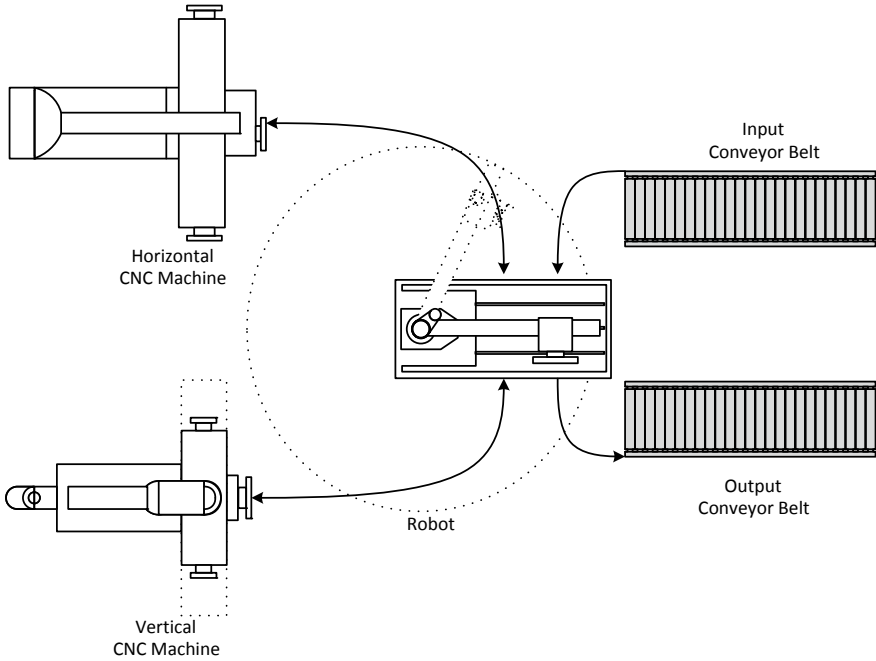


Fig. 2 A cyclic production system

2. The robot moves a raw part from the conveyor and loads it at the horizontal machine.
3. The milling operations are performed at the horizontal machine while the robot backs off (returns).
4. The robot unloads the semi-finished part from the horizontal machine, loads it to the vertical machine and then it returns.
5. The drilling operation is performed at the vertical machine, and simultaneously the robot performs step 2.
6. The robot unloads the finished part from the vertical machine, deposits it on the conveyor and returns.

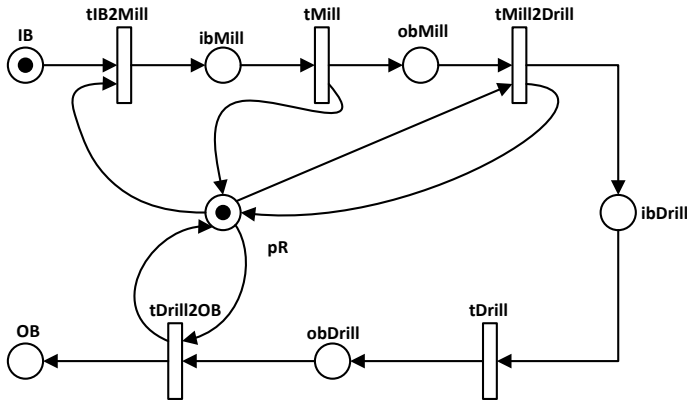
In normal operations, the steps 2–6 are repeated.

### The Petri Net Model

Figure 3 shows the Petri Net model. Table 1 presents the elements (transitions and places) involved in the model. Figure 4 shows the reachability graph when the system works on two input materials.

### GPenSIM Implementation of the Petri Net

The two files MSF and PDF are given below.



**Fig. 3** The Petri net model

**Table 1** The elements of the Petri net model

Place	Transition
IB: the input buffer for raw material	–
pR: availability of the robot	tIB2Mill: the activity of robot moving raw material from IB into ibMill
ibMill: the input buffer of the milling machine	tMill: milling operation
obMill: the out buffer of the milling machine	tIB2Mill: the activity of robot moving semi-product from obMill to ibDrill
ibDrill: the input buffer of the drilling machine	tDrill: drilling operation
obDrill: the output buffer of the drilling machine	tDrill2OB: the activity of robot moving completed product from obDrill to OB
OB: the output buffer for the products	–

**MSF:**

```

% Repetitive Production System
% MSF: the main file to run simulation
global global_info
png = pnstruct('PDF_rps');
dyn.m0 = {'IB',2, 'Robot',1};
dyn.ft ={'allothers', 2};
pni = initialdynamics(png, dyn);
cotree(pni, 1, 0);
    
```

**PDF:**

```

function [png] = PDF_rps()
% file: PDF_rps.m:
% PDF for "Repetitive Production System"
    
```

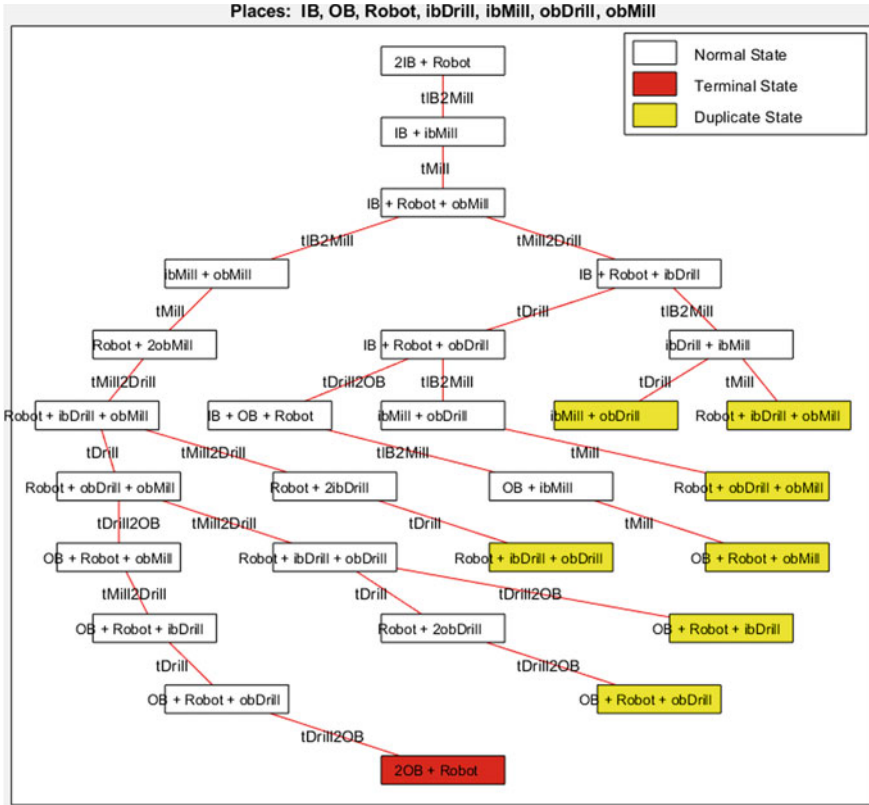


Fig. 4 Reachability graph generated from the Petri net model

```

png.PN_name = 'Repetitive Production System';
png.set_of_Ps = {'IB', 'ibMill', 'obMill', ...
'ibDrill', 'obDrill', 'OB', 'Robot'};
png.set_of_Ts = {'tIB2Mill', 'tMill', 'tMill2Drill',
'tDrill', 'tDrill2OB'};
png.set_of_As = {...
'IB', 'tIB2Mill', 1, 'Robot', 'tIB2Mill', 1, ... %tIB2Mill
'tIB2Mill', 'ibMill', 1, ... % tIB2Mill
'ibMill', 'tMill', 1, 'tMill', 'Robot', 1, 'tMill',
'obMill', 1, ... % tMill
'obMill', 'tMill2Drill', 1, 'Robot', 'tMill2Drill',
1, ... % tMill2Drill
'tMill2Drill', 'ibDrill', 1, 'tMill2Drill', 'Robot',
1, ... % tMill2Drill
'ibDrill', 'tDrill', 1, 'tDrill', 'obDrill',
1, ... % tDrill
    
```

```

'obDrill','tDrill2OB',1, 'Robot','tDrill2OB',
1,... .. % tDrill
'tDrill2OB','Robot',1, 'tDrill2OB','OB',
1, ... % tDrill2OB
};

```

### Model Checking on the Petri Net Model

Since the robot is involved in four different activities involving the two machines, a mishap can happen. For example, the robot places the part produced by the drilling machine into the input buffer of the milling machine, instead of the output buffer OB. Hence, the model is checked for the following properties, generally known as the safeness properties:

- Property-1: only the raw input materials enter the input buffer of the milling machine.
- Property-2: after the milling operation, the robot transport the part from the out buffer of the milling machine immediately into the input buffer of the drilling machine.
- Property-3: after the drilling operation, the robot moves the part right into the output buffer OB.

Using CTL, these properties can be formally specified as follows:

- Property-1:  $IB \ U \ ibMill$
- Property-2:  $tMill \longrightarrow obMill \ U \ ibDrill$
- Property-3:  $tDrill2OB \longrightarrow N \ OB$

It can be easily verified that all these properties are satisfied by the reachability graph shown in Fig. 4. For example, Fig. 5 shows that the property 3 being satisfied; in Fig. 5, the operation tDrill2OB (marked with continuous red ovals) is immediately followed by states that show the part in output buffer OB (marked with broken green ovals).

## 5 Remarks and Conclusions

Activity-oriented Petri Nets (AOPN) is a discrete-event modeling language that is an extension of the Resource-Oriented Petri Net. AOPN with which resources can be abstracted away from the Petri net model allows to significantly reduce the complexity and size of Petri network models. This is particularly important in the context of using AOPN implementation as an effective and efficient tool for modeling and analysis of discrete production systems, for which models, the use of standard Petri net models significantly increases their complexity. The chapter presents a computer implementation of the AOPN concept in the form of GPenSIM software on the MATLAB platform. An incorporating Automatic Model Checking into GPenSIM





3. Banaszak, Z.A., Polak, M.: Deadlock-free distributed control for repetitive flows. In: Sixth International Workshop on Discrete Event Systems, Proceedings, Zaragoza, Spain, 4 October 2002, pp. 273–278. IEEE (2002)
4. Bocewicz, G., Nielsen, P., Banaszak, Z.A., Dang, V.Q.: Cyclic steady state refinement: multimodal processes perspective. In: Frick, J., Laugen, B.T. (eds.) *Advances in Production Management Systems. Value Networks: Innovation, Technologies, and Management*. APMS 2011, pp. 18–26. Springer, Berlin (2012)
5. Bocewicz, G., Wójcik, R., Banaszak, Z.A., Pawlewski, P.: Multimodal processes rescheduling: cyclic steady states space approach. *Math. Probl. Eng.* **2013** (2013)
6. Bocewicz, G., Janardhanan, M.N., Krenczyk, D., Banaszak, Z.: Traffic flow routing and scheduling in a food supply network. *Ind. Manag. Data Syst.* **117**(9), 1972–1994 (2017)
7. Cameron, A., Stumptner, M., Nandagopal, N., Mayer, W., Mansell, T.: Rule-based peer-to-peer framework for decentralised real-time service oriented architectures. *Sci. Comput. Program.* **97**, 202–234 (2015)
8. Chang, H.: A method of gameplay analysis by Petri net model simulation. *J. Korea Game Soc.* **15**, 49–56 (2015)
9. Cheng, A., Christensen, S., Mortensen, K.: Model checking coloured Petri nets – exploiting strongly connected components. *DAIMI Rep. Ser.* **26**, 519 (1997)
10. Davidrajuh, R.: ACTivity-oriented Petri net for scheduling of resources. In: 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Seoul, South Korea, 14–17 October 2012, pp. 1201–1206. IEEE (2012)
11. Davidrajuh, R.: Developing a Petri nets based real-time control simulator. *Int. J. Simul. Syst. Sci. Technol.* **12**(3), 28–36 (2012)
12. Davidrajuh, R.: Outperforming genetic algorithm with a brute force approach based on activity-oriented Petri nets. In: Graña, M., López-Guede, J., Etxaniz, O., Herrero, Á., Quintián, H., Corchado, E. (eds.) *International Joint Conference SOCO-16-CISIS-16-ICEUTE-16. SOCO 2016, ICEUTE 2016, CISIS 2016*, pp. 454–463 (2017)
13. Davidrajuh, R.: Activity-oriented Petri nets: aligning real-world buffers with virtual places. *Int. J. Simul. Syst. Sci. Technol.* **18**(3), 7.1–7.6 (2017)
14. Davidrajuh, R.: *Modeling Discrete-Event Systems with GPenSIM*. Springer, Cham (2018)
15. GPenSIM: a general purpose Petri net simulator. <http://www.davidrajuh.net/gpensim> (2019). Accessed 15 Jan 2019
16. Hillion, H.P., Proth, J.M.: Performance evaluation of job-shop systems using timed event graphs. *IEEE Trans. Autom. Control* **34**(1), 3–9 (1989)
17. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri nets and CPN tools for modelling and validation of concurrent systems. *Int. J. Softw. Tools Technol. Transf.* **9**(3–4), 213–254 (2007)
18. Jones, N.D., Landweber, L., Lien, Y.E.: Complexity of some problems in Petri nets. *Theor. Comput. Sci.* **4**(3), 277–299 (1977)
19. Jyothi, S.D.: Scheduling flexible manufacturing system using Petri-nets and genetic algorithm. Technical report, Department of Aerospace Engineering, Indian Institute of Space Science and Technology, Thiruvananthapuram, India (2012)
20. Korbaa, O., Camus, H., Gentina, J.C.: FMS cyclic scheduling with overlapping production cycles. In: Proceedings of the International Conference on Application and Theory of Petri Nets, Workshop on Manufacturing and Petri Nets, pp. 35–52 (1997)
21. Korbaa, O., Camus, H., Gentina, J.C.: A new cyclic scheduling algorithm for flexible manufacturing systems. *Int. J. Flex. Manuf. Syst.* **14**(2), 173–187 (2002)
22. Murata, T.: Petri nets: properties, analysis and applications. *Proc. IEEE* **77**(4), 541–580 (1989)
23. Mutarraf, U., Barkaoui, K., Li, Z., Wu, N., Qu, T.: Transformation of business process model and notation models onto Petri nets and their analysis. *Adv. Mech. Eng.* **10**(12), 1–21 (2018)
24. Ohl, H., Camus, H., Castelain, E., Gentina, J.C.: A heuristic algorithm for the computation of cyclic schedules and the necessary WIP to obtain optimum cycle time. In: Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology, Troy, NY, USA, 10–12 October 1994, pp. 339–344. IEEE (1994)

25. Szpyrka, M.: Petri Nets in Modeling and Analysis of Concurrent Systems (in Polish). WNT, Warszawa (2008)
26. Valentin, C.: Modeling and analysis methods for a class of hybrid dynamic systems. In: Proceedings of ADPM '94, Brussels, Belgium, pp. 221–226 (1994)
27. Wójcik, R.: Constraint programming approach to designing conflict-free schedules for repetitive manufacturing processes. In: Cunha, P.F., Maropoulos, P.G. (eds.) Digital Enterprise Technology, pp. 267–274. Springer, Boston (2007)
28. Wójcik, R.: Designing a no-wait cyclic schedule for a class of concurrent repetitive production processes. IFAC-PapersOnLine **51**(11), 1305–1310 (2018). <https://doi.org/10.1016/j.ifacol.2018.08.352>

# Index

## A

Analysis problem, 111, 121  
Asap schedule, 10  
Automated Guided Vehicle (AGV), 137

## B

Brute-force, 62

## C

Cellular manufacturing, 64  
Constraint satisfaction problem, 119, 124, 126  
    recursive, 126  
Continuous Time System (CTS), 88  
Cycle time, 32, 33, 66  
    average, 10  
Cyclic Assignment Problem in Production Cell (CAPPC), 66  
    in non-permutational FSP, 68  
Cyclic flow shop robotic cell, 64  
Cyclic production, 176, 185  
    scheduling, 176, 177  
Cyclic steady state, 131

## D

Delivery operation, 112, 120  
Deterministic Finite State Automaton (DFSA), 89, 93  
Discrete Event System (DES), 88  
Docking station, 112  
Dynamic Programming (DP), 32

## F

Fitness Landscape (FL), 65  
Fitness Landscape Analysis (FLA), 64  
Flexible Assembly System (FAS), 88  
Flow Shop Problem (FSP), 32, 64, 68  
    2-machine, 42

## G

GPenSIM, 178, 180, 181, 184  
Graph, 38, 40, 47

## I

Iterated Local Search (ILS), 46

## K

K-periodic schedule, 10

## L

Local Optima Network (LON), 64  
    measure, 73  
        assortativity, 73  
        average shortest path to optimum, 74  
        global clustering, 74  
        minimum as percent, 74  
Local Search (LS), 46

## M

Manufacturing Execution Systems (MES), 138

Milk-run system, 107, 111, 112, 119, 126  
 Minimal cycle time, 67  
 Minimal Part Set (MPS), 32, 66  
 MIP, 64  
 MMRS, 88  
   admissible-event function, 96  
   deadlock avoidance, 95  
   feasible-event function, 96  
   motion space tessellation, 90  
   Multiple Mobile Robot System, 89  
   Priority control, 101  
   RAS-controller, 100  
   RAS-model, 93  
   Safety problem, 96  
   supervisor, 96  
 Model  
   graph, 37  
   mathematical, 35  
 Model checking, 178, 180, 184  
 Multimodal processes, 177  
 Multiple Mobile Robot System, 88

**N**

Normalised Synchronous DataFlow Graph,  
 8

**P**

Permutation, 33, 37, 42  
   inverse, 36  
   natural, 45  
 Petri Net, 176–178, 184  
   activity-oriented, 178, 184  
 Pick-up operation, 112, 115, 120

Production takt time, 107

**R**

Reachability problem, 177, 178  
 Reference model, 106  
 Resource Allocation System (RAS), 89, 93  
   FREE-RANGE\*-RAS, 93  
   FREE-RANGE-k-RAS, 93  
   FREE-RANGE-RAS, 93  
 Rhythmic production, 176

**S**

Schedule, 32, 33, 35, 66  
 Setup, 31  
   disjoint, 42  
   sequence-dependent, 32  
   team, 32, 33  
     route, 34, 35  
 Strictly periodic schedule, 15  
 Synchronous DataFlow Graph, 3–5  
 Synthesis problem, 111, 122

**T**

Throughput of a SDF, 9  
 Traveling Salesman Problem (TSP), 31  
 Tugger train, 106, 112

**U**

Uniform precedence graph, 8  
 Useful initial marking, 7